

Chi N. Thai

Exploring Robotics with ROBOTIS Systems



 Springer

Exploring Robotics with ROBOTIS Systems

Chi N. Thai

Exploring Robotics with ROBOTIS Systems

Chi N. Thai
College of Engineering
University of Georgia
Athens, GA, USA

ISBN 978-3-319-20417-8 ISBN 978-3-319-20418-5 (eBook)
DOI 10.1007/978-3-319-20418-5

Library of Congress Control Number: 2015944075

Springer Cham Heidelberg New York Dordrecht London
© Springer International Publishing Switzerland 2015

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made.

Printed on acid-free paper

Springer International Publishing AG Switzerland is part of Springer Science+Business Media
(www.springer.com)

*To my parents for giving me life
and to my wife Christine and daughter
Emily for their enduring love and support.*

Contents

1	Motivations and Instructional Approach.....	1
1.1	Motivations	1
1.2	Instructional Approach	2
	References	3
2	ROBOTIS' Robot Systems	5
2.1	General Systems Description.....	5
2.2	Robotics Kits Considered in Book.....	9
2.3	Micom Training Kit	14
2.4	System(s) Selection Criteria	15
2.5	Review Questions for Chap. 2	16
3	Hardware Characteristics	19
3.1	The Atmel AVR Family	19
3.1.1	CM-5 (Discontinued)	19
3.1.2	CM-510 (Discontinued).....	23
3.2	The STM ARM Cortex M3 Family	25
3.2.1	CM-530	25
3.2.2	CM-900 (Discontinued).....	26
3.2.3	OpenCM-9.04 Series.....	27
3.3	The Dynamixel Actuators Family	31
3.3.1	The AX Series.....	31
3.3.2	The MX Series	33
3.4	ROBOTIS Sensors Family	35
3.4.1	AX-S1 and IRSA	36
3.4.2	AX-S20 (Discontinued).....	37
3.4.3	Foot Pressure Sensor (FPS: From HUV Robotics).....	39
3.4.4	HaViMo 2.0.....	40
3.4.5	GPIO (5-Pin) DMS Sensor	40
3.4.6	GPIO (5-Pin) Gyroscope Sensor GS-12	41

3.4.7	Other GPIO (5-Pin) Sensors and Output Devices.....	41
3.4.8	Recent Adaptations of Smart Phone Features.....	42
3.5	Review Questions for Chap. 3	42
	References	45
4	Software Tools	47
4.1	Dynamixel Wizard Tool.....	49
4.1.1	TTL (3-Pin) and RS-485 (4-Pin) Dynamixels	49
4.1.2	XL-TTL (3-Pin) Dynamixels	50
4.1.2.1	OpenCM-9.04-C	51
4.1.2.2	OpenCM-9.04-B.....	52
4.2	Manager Tool	52
4.2.1	CM-5, CM-510, CM-530.....	52
4.2.2	OpenCM-9.04-A/B/C.....	53
4.3	Task Tool.....	54
4.3.1	CM-5, CM-510, CM-530.....	55
4.3.2	CM-9.04-C	56
4.4	Motion Tools (V.1 and V.2).....	57
4.5	R+ Design Tool	59
4.6	“If I Were to Restart ...”	60
4.7	Review Questions for Chap. 4	60
5	Foundational Concepts.....	61
5.1	“Sense-Think-Act” Paradigm	61
5.2	Primer for MANAGER and TASK Tools	65
5.2.1	MANAGER Capabilities.....	65
5.2.2	Basic TASK Usage.....	70
5.3	“Sequence Commander” Project	70
5.4	“Smart Avoider” Project	72
5.5	“Line Tracer” Project.....	75
5.5.1	Mechanical Design Features.....	76
5.5.2	IR Array Sensor (IRSA).....	77
5.5.3	Programming Maneuvers for Line Tracer.....	77
5.6	“Remote Controlled CarBot” Project	79
5.7	Review Questions for Chap. 5	83
5.8	Review Exercises for Chap. 5	84
	References	85
6	Actuator Position Control Basics.....	87
6.1	AX-12/18 Position Control with TASK.....	87
6.2	Using Motion Editor (V.1)	92
6.2.1	Characteristics of a Motion Page in Motion V.1	92
6.2.2	Application to a GERWALK Robot.....	95
6.3	Form and Function of Walking Robots.....	95
6.4	Review Questions for Chap. 6	100

6.5	Review Exercises for Chap. 6	101
	References	102
7	Advanced Position Control	103
7.1	“Torque” Effects	103
7.1.1	Torque Limit, Present Position and Present Load	104
7.1.2	Adjusting Torque Limit Dynamically	104
7.2	“Joint Offset” Effects	106
7.3	A Load Sensing Gripper	107
7.4	Review Questions for Chap. 7	108
7.5	Review Exercises for Chap. 7	108
	References	109
8	Wireless Communications Programming	111
8.1	ZigBee Broadcast Channel Differences	112
8.2	Broadcast Use of RC-100 (NIR and ZigBee)	113
8.3	Message “Shaping” Concepts	115
8.3.1	Mimicking Grippers	115
8.3.2	Leader-Follower GERWALKS	118
8.3.3	Multiple Users and Multiple Robots (ZigBee Only)	119
8.4	PC to Robots Communications via C/C++	122
8.5	ZigBee and Bluetooth Performances	123
8.6	Review Questions for Chap. 8	123
8.7	Review Exercises for Chap. 8	125
	References	126
9	Advanced Sensors	127
9.1	Humanoid Static Balance with AX-S20	127
9.1.1	2-Leg Static Balance with AX-S20	128
9.1.2	1-Leg Static Balance	132
9.2	Humanoid Dynamic Balance with GS-12	133
9.2.1	Walk Enhancement with GS-12	135
9.2.2	Fall Detection with GS-12	136
9.3	Humanoid Balance with FPS	136
9.3.1	FPS Data Acquisition	136
9.3.2	Humanoid 1-Leg Balance with FPS	138
9.4	HaViMo2 Applications	139
9.4.1	HaViMo2 Features and Usage	139
9.4.2	HaViMo2 Application to a CM-5 CarBot	140
9.5	Review Questions for Chap. 9	144
9.6	Review Exercises for Chap. 9	144
	References	145
10	Embedded C Options	147
10.1	Embedded C vs. RoboPlus’ TASK	148
10.2	Embedded C for the OpenCM-9.00/9.04	151
10.3	Embedded C for the CM-510 and CM-530	152
10.3.1	Tutorials for CM-510	152

10.3.2	Tutorials for CM-530.....	153
10.3.3	Future Support for Embedded C for CM-510/530?	153
10.4	Motion Programming and Embedded C.....	154
10.5	Review Questions for Chap. 10.....	155
10.6	Review Exercises for Chap. 10	155
	Reference	155
11	ROBOTIS-MINI System	157
11.1	PC to MINI Wireless Options	158
11.2	New Motion Concepts in MOTION V.2	159
11.2.1	Unlimited File Size for MTNX	159
11.2.2	Efficient Motion Data Structure	160
11.3	PC Control of Robot Moves.....	163
11.4	Synchronizing LEDs to Motion	165
11.5	Fight Choreography for two MINIs via ZigBee.....	167
11.6	Fight Choreography for two MINIs via BlueTooth.....	169
11.7	Review Questions for Chap. 11	171
11.8	Review Exercises for Chap. 11	171
	References.....	172

Chapter 1

Motivations and Instructional Approach

1.1 Motivations

My personal robotics journey began with RoboCup 2007 hosted by Ga Tech that summer. I was so impressed by the advancement in humanoid robotics on display there and thought that this could be an important component for engineering instruction at The University of Georgia, as at that time we were seeking to expand engineering beyond biological and agricultural engineering. Also at RoboCup 2007, I met Mr. Jinwook Kim from ROBOTIS and was introduced to the Bioloid Comprehensive and Expert kits which were about the only “in-depth” but “affordable” robotic systems at the time.

In 2008 when I started to draft out the instructional materials for my “Embedded Robotics” course, the only source of ROBOTIS technical information was from their “Quick Start”, “Programming Guide” and “Expert” manuals. I was also much inspired by the works of Bekey (2005), Bräunl (2006), Mataric (2007) and Miller et al. (2008) which resulted in my adoption of a project-based approach to my course. Also during this time, I was going through some trainings by the UGA Center for Teaching and Learning which made me aware of “learner-centered” approaches (Fink 2003; Weimer 2002) and screen/classroom capture technologies such as Camtasia Studio (Thai et al. 2008). I also had found the “Spiral” model from Dr. Joseph Bergin (Bergin 2012) very inspiring in designing the flow of presented topics and instructional materials in the “Embedded Robotics” course and also in this book. Unfortunately, not all classroom teaching approaches can be directly transferred into a “book”, for example I modified the “contract” teaching style (Wikipedia 2013) by allowing students to choose their own projects to suit their own interests and personal strengths (Thai 2014). However, in this book, I can only describe those projects and provide web links to show the students achievements.

I was also fortunate to have help from outstanding students during the development period of this course:

- Mr. Alex Fouraker (pursuing his BSAE degree) was instrumental in exploring the limits of what the Bioloid kits can do: Alex’s work on the GERWALK climbing

stairs is still the most accessed item at the ROBOTIS Gallery web site with 255,192 views as of 12/10/2014 (Thai 2009).

- In Spring 2010, this course was taught for the first time and more instructional design details for this particular implementation can be found in this article (Thai 2010). During this course, Mr. Matthew Paulishen (also pursuing his BSAE degree) had extended many BIOLOID projects for PC-side Visual C++ and Machine Vision programming for bipedal robots (Thai 2011). Since then this course had been fine-tuned using student feedbacks and in accordance with the continuing hardware and software updates from ROBOTIS.

Between Spring 2012 and Fall 2013, I had the opportunity to repurpose my UGA classroom recordings with Drs. Yan-Fu Kuo and Ping-Lang Yen from National Taiwan University for a Distance Education project based on a Flipped Classroom model (Thai et al. 2013). This project has helped in refining the scaffolding of the presented materials and genesis of new course projects.

From their end, ROBOTIS had their Q&A web site since 2007 and added more web resources for users such as their e-Manual (<http://support.robotis.com/>) in early 2010 and RobotSource (http://www.robotsource.org/?na=en&pc_ver=1) in late 2011. Furthermore, ROBOTIS continues to improve the instructional design of their paper-based manuals that are included in their BIOLOID STEM and PREMIUM robotic kits. At present, there are also several web-based Forums such as the ones from RoboSavvy (<https://robosavvy.com/forum/viewforum.php?f=5>) and Trossen Robotics (<http://forums.trossenrobotics.com/>) providing ROBOTIS user communities worldwide and they perform many useful functions. However, a reading of the postings from the above web sites would show that some beginners are still having a tough time learning and some folks are still having problems finding solutions to suit their needs.

Those were my motivations to write this book using a web-based multimedia approach for illustrating robotics concepts, code implementations and actual resulting robot behaviors. This approach also allows the “thinning” the book (a video is worth many pages of text!), for easier updates and for users to skip subjects they do not need. This book is mainly designed for self-learners and presents in a more organized manner various information sources from ROBOTIS technical training manuals (paper-based and web-based).

1.2 Instructional Approach

In keeping with the “Spiral” instructional model and a project-based approach, the rest of this book is structured as follows:

- Chapter 2—Overall view of ROBOTIS robotics systems from 1999 to 2014.
- Chapter 3—Hardware characteristics of the robot systems used in this book (i.e., BIOLOID and Open-CM): controllers, actuators and sensors.
- Chapter 4—Description of the evolution of ROBOTIS software tools since 1999 and illustration of their basic use for firmware updating and evaluation of wired and wireless control performances.

- Chapter 5—Foundational robotics concepts using wheeled robots and the Manager and Task tools: basic use of actuators and sensors, autonomous behaviors achieved via reaction control and behavior control approaches, introductory remote communication and control concepts. Projects implemented: Sequence Commander, Line Tracer and Smart Avoider.
- Chapter 6—Multi-link robots and introductory actuator position control concepts using the Task and Motion V.1 tools: Proportional-Integral (PI) implementations, Motion Pages, and required modifications to remote control programming application. Projects implemented: BugBot, Bipedal Bots.
- Chapter 7—Applications of advanced position control features such as ‘Load Limit’, ‘Present Load’ and ‘Join Offset’, Callback functions. Projects implemented: GERWALK and Load-sensing gripper.
- Chapter 8—Advanced communications programming concepts via ZigBee: 1-to-1 and broadcast modes. Projects implemented: Embedding special signals into standard packets for bot-to-bot and PC-to-bots application programming, Leader and Follower grippers, Multiple-users control scheme.
- Chapter 9—Integrating advanced sensors such as gyroscope, inertial measuring unit, foot pressure sensor, and color video camera. Projects implemented: Balance of Humanoid Robot, Color Object Search by Carbot and GERWALK.
- Chapter 10—Embedded C options and implementations on BIOLOID and OpenCM-9.04-A/B systems using ROBOTIS SDK and OpenCM IDE examples.
- Chapter 11—ROBOTIS(DARWIN)-MINI system: PC wireless communications options, new motion concepts in Motion V.2, sensor integration with TASK & MOTION, and choreography application to two MINIs.

In closing for this chapter, I would like to quote Dr. Ben Shneiderman from his book “Leonardo’s Laptop” (2002, p. 113):

... we might rethink education in terms of collect-relate-create-donate:
 COLLECT—Gather information and acquired resources
 RELATE—Work in collaborative teams
 CREATE—Develop ambitious projects
 DONATE—Produce results that are meaningful to someone outside the classroom ...

References

- Bekey GA (2005) Autonomous robots. MIT, Cambridge
- Bergin J (2012) Pedagogical patterns: advice for educators. CreateSpace Independent Publishing Platform, p 115
- Bräunl T (2006) Embedded robotics. Springer, Heidelberg
- Fink LD (2003) Creating significant learning experiences. Jossey-Bass, San Francisco
- Matarić MJ (2007) The robotics primer. MIT, Cambridge
- Miller DP et al (2008) Robots for education. In: Siciliano B, Khatib O (eds) Springer handbook of robotics. Springer, Heidelberg, pp 1283–1301
- Shneiderman B (2002) Leonardo’s laptop. MIT, Cambridge, pp 113–114
- Thai CN (2009) Bioloid GERWALK robot going up stairs steps. <http://thai.engr.uga.edu/Bioloid/Stairs/index.html>. Accessed 13 Dec 2014

- Thai CN (2010) Teaching robotics to students with mixed interests. http://thai.engr.uga.edu/PDF/EIAE_10_Teaching_Robotics.pdf. Accessed 13 Dec 2014
- Thai CN (2011) Biped robot going up stairs steps. http://thai.engr.uga.edu/RobotVids/BipedUpStairs_1.wmv. Accessed 13 Dec 2014
- Thai CN (2014) Syllabus CSEE/ENGR-4310: Embedded Robotics (Fall 2014). <http://thai.engr.uga.edu/PDF/CSEE-ENGR-4310.pdf>. Accessed 13 Dec 2014
- Thai CN et al (2008) Robotics-based curriculum development for an immigration course into computer systems engineering. http://thai.engr.uga.edu/PDF/Thai_EIAE_08.pdf. Accessed 13 Dec 2014
- Thai CN et al (2013) Cooperative teaching in a distance education environment. <http://thai.engr.uga.edu/PDF/CooperativeTeachingDEEnvironment.pdf>. Accessed 13 Dec 2014
- Weimer M (2002) Learner-centered teaching. Jossey-Bass, San Francisco
- Wikipedia (2013) Student-teacher contract (teaching style). [http://en.wikipedia.org/wiki/Student-teacher_contract_\(teaching_style\)](http://en.wikipedia.org/wiki/Student-teacher_contract_(teaching_style)). Accessed 13 Dec 2014

Chapter 2

ROBOTIS' Robot Systems

2.1 General Systems Description

ROBOTIS Co., Ltd (Seoul, South Korea) was founded by Bill Byoung-Soo Kim in 1999 along with two other engineers. The current CEO (Byoung-Soo Kim) and Vice-President (In-Yong Ha) were from the original team. The ROBOTIS name derived from the question “What is a robot?” and their vision statement can be read at http://en.robotis.com/index/company_01.php. In 2009, ROBOTIS opened their USA office in Irvine, California and currently ROBOTIS has more than 200 partners in 40 countries worldwide. In a 2014 interview with Robot Magazine, CEO Kim shared his strategy for future products (http://en.robotis.com/BlueAD/board.php?bbs_id=news&mode=view&bbs_no=797584&page=3&key=&keyword=).

“Dynamixel” is the brand name uniquely connected to ROBOTIS. “Dynamixel” encapsulates several modularization and standardization concepts applied to both sensors and actuators equipped with embedded computing and communications capabilities (Fig. 2.1).

In 1999, ROBOTIS launched their first product called Didi and Titi. This link (http://en.channel.pandora.tv/channel/video.ptv?c1=05&ch_userid=do7minate&prgid=49759295&ref=na) shows a TV commercial for Didi and Titi (Fig. 2.2).

Since then, ROBOTIS has released 20 more products:

1. Toma (2002—released in Korea only).
2. Dynamixel—AX-12 (2003).
3. Cycloid (2004—released in Korea only).
4. BIOLOID—Beginner and Comprehensive (2005).
5. UR1A (2006—released in Korea only).
6. Dynamixel—RX-64 (2006).
7. OLLO (2008).
8. Dynamixel—EX-106 (2008).
9. BIOLOID PREMIUM (2009).
10. Dynamixel—MX series (2011).

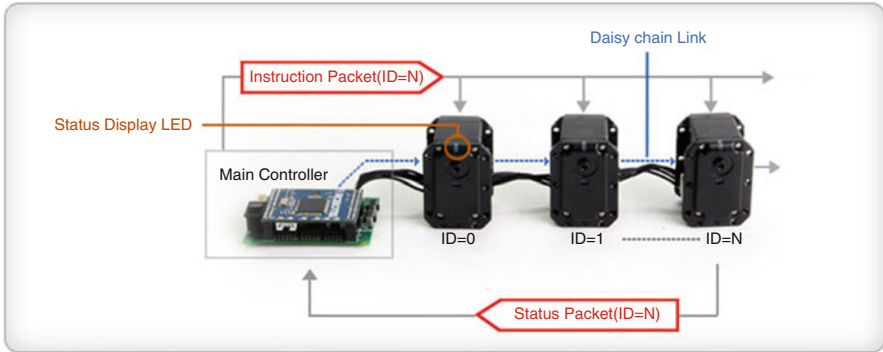


Fig. 2.1 “Dynamixel” concept

Fig. 2.2 Didi and Titi



11. DARWIN-OP (2011), renamed ROBOTIS-OP in 11/2014.
12. BIOLOID STEM (2012).
13. IDEAS (2013).
14. THOR-MANG (2013).
15. Dynamixel-Pro H-series (2013).
16. Dynamixel—XL-320 (2014).
17. DARWIN-MINI (5/2014), renamed ROBOTIS-MINI in 11/2014.
18. Dynamixel-Pro L-series (2014).
19. DREAM (2014).
20. SMART (~2015).

This list shows ROBOTIS' commitment to continuing development and improvement and to serve a very broad clientele in terms of age as well as technical level. These products had been adopted by hobbyists of all ages as well as teachers and researchers worldwide (please visit <http://www.ROBOTIS.com> for more details).

The systems designed for young children are: OLLO, IDEAS, DREAM and SMART. They are colorful and use a quick-connect system adapted from the

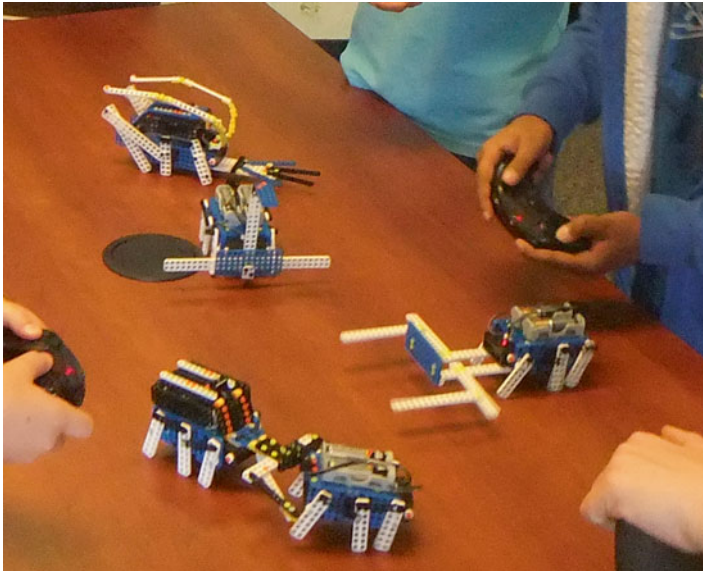


Fig. 2.3 OLLO robots in action

standard rivet concept to ease hands-on creative activities for children. They can be constructed to be simple mechanical assemblies (IDEAS) as well as programmable and motorized robots (OLLO, DREAM). Their most recent SMART system is designed to be used with mobile devices such as smart phones and tablets. Their embedded controllers are based on the Atmel AVR chip with more recent systems such as DREAM and SMART using the STM32F103C8 or STM32L151C8 from STMicroelectronics (Fig. 2.3).

The BIOLOID systems (BEGINNER, COMPREHENSIVE, STEM and PREMIUM) are designed to be various entry points (depending on one's budget) into the robotics field for those interested in taking a more comprehensive journey into this knowledge area. They use standard screws and nuts for a sturdier fastening method, with some parts using the OLLO's rivet system also. The older kits (BEGINNER and COMPREHENSIVE) were designed off the Atmel AVR chip while more recent systems (STEM and PREMIUM) rely on the ARM architecture provided such embedded controllers as STM32F103C8 and STM32F103RE from STMicroelectronics (Fig. 2.4).

In 2012, ROBOTIS made a strategic shift into the open hardware-software movement with their OpenCM systems whereas users can collaborate on hardware and software development worldwide (http://www.robotsource.org/bs/bd.php?bt=forum_CM9DeveloperWorld) (Fig. 2.5).

ROBOTIS is also well known for its humanoid robots in the competitive and research arenas such as the GP (http://www.ROBOTIS.com/xe/BIOLOID_GP_en), ROBOTIS(DARWIN)-OP (http://www.ROBOTIS.com/xe/darwin_en), and THOR-MANG (http://www.youtube.com/watch?v=j_YSeOa9yAQ). In Spring 2014,

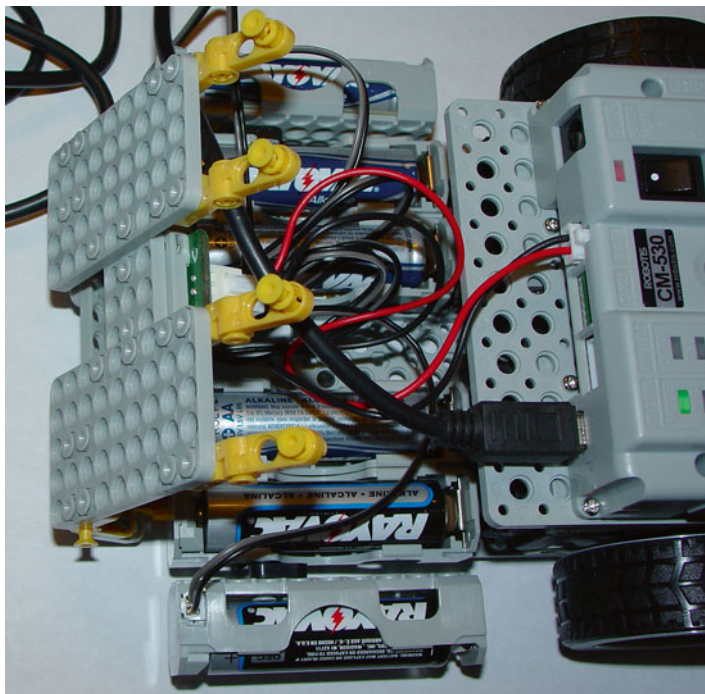


Fig. 2.4 A Bioloid STEM robot

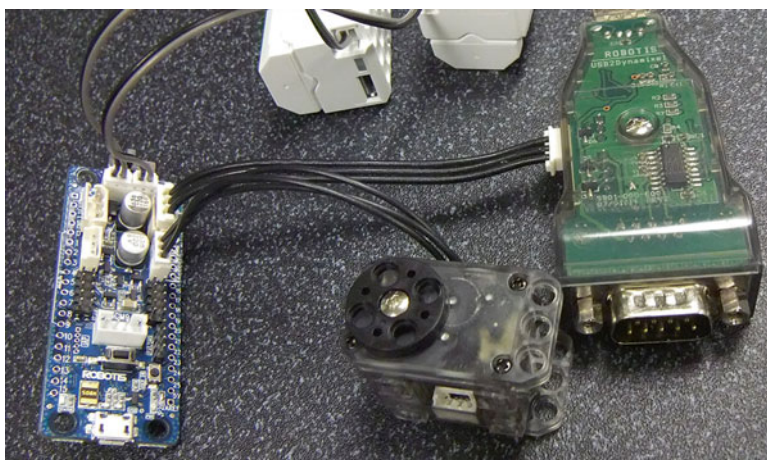


Fig. 2.5 OpenCM-9.04-B controller (*left*), XL-320 servo motor (*center*), and USB2Dynamixel communication converter (*right*)

ROBOTIS released a more affordable humanoid robot called ROBOTIS(DARWIN)-MINI (http://www.ROBOTIS.com/xe/ROBOTIS_DARWIN_MINI_en) based on the OpenCM9.04-C controller and the XL-320 servo motor. Outwardly, the DARWIN-MINI is roughly a half-scale version of the DARWIN-OP, but with



Fig. 2.6 ROBOTIS(DARWIN)-MINI action figure

reduced capabilities and a different kinematic linkage solution for its legs. It is designed to be operated and programmable from mobile devices as well as personal computers (MS Windows) (Fig. 2.6).

As this book is geared towards university undergraduate students or self-learners, only the following systems will be considered in further details: BIOLOID BEGINNER, COMPREHENSIVE, STEM and PREMIUM, OpenCM and ROBOTIS(DARWIN)-MINI.

2.2 Robotics Kits Considered in Book

The BIOLOID BEGINNER and COMPREHENSIVE systems use the CM-5 as its main controller which is an Atmel ATmega @ 16 MHz and with 128 KB of flash memory. It connects to ROBOTIS' sensors and actuators via the Dynamixel TTL bus (3-pin) and has ZigBee wireless communications capabilities via the ZIG-100 modules (Fig. 2.7).

Actually there was another CM-5 based system called BIOLOID EXPERT that was available between 2005 and 2009. It used the same basic hardware as the COMPREHENSIVE, but had a wireless color video camera and a Visual C++ V.6

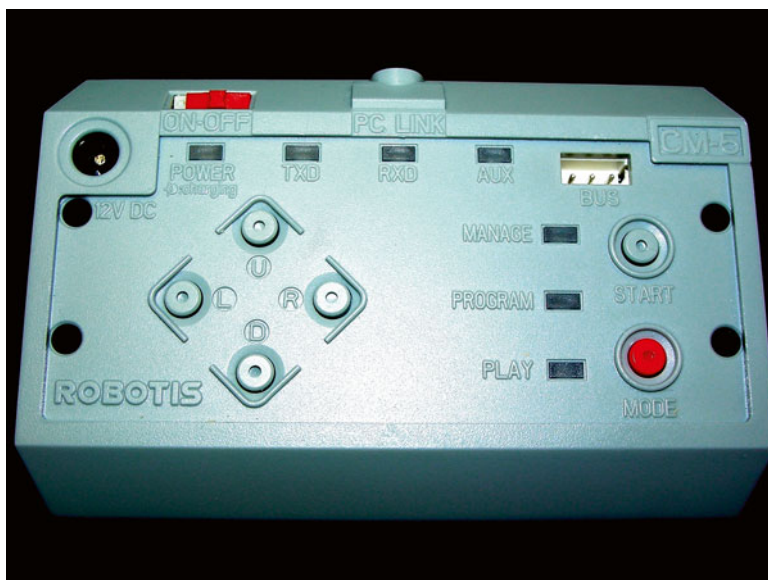
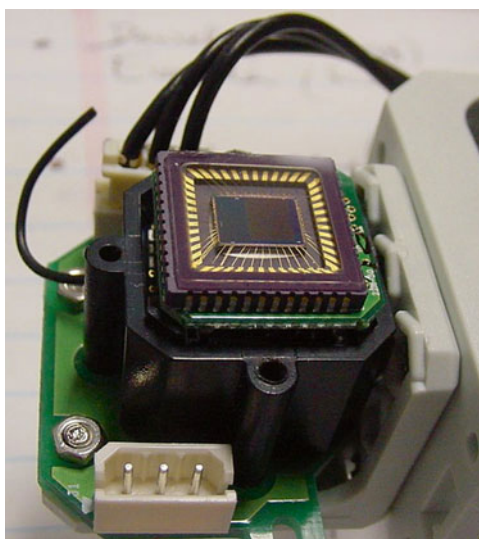


Fig. 2.7 CM-5 Controller (discontinued)

Fig. 2.8 Wireless color video camera from CM-5 EXPERT kit (discontinued)



library with functions to control sensors, actuators, video camera and also to perform machine vision processing routines and ZigBee wireless communications (Fig. 2.8).

When the BIOLOID PREMIUM system first came out in 2009, it was shipped with the CM-510 controller which was a 16-MHz ATmega with 256 KB flash



Fig. 2.9 CM-510 (discontinued) and CM-530 controllers

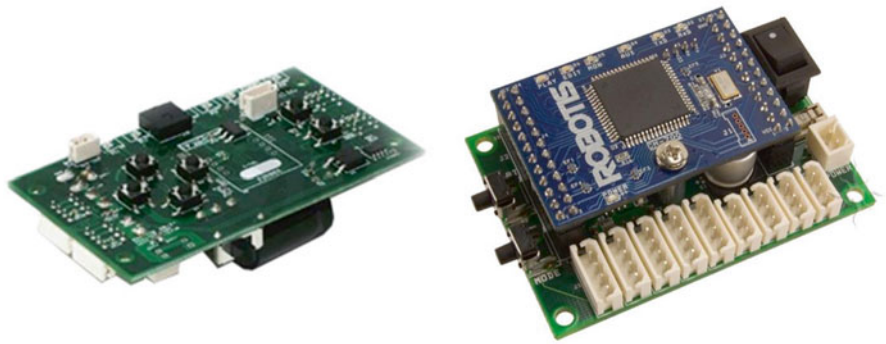


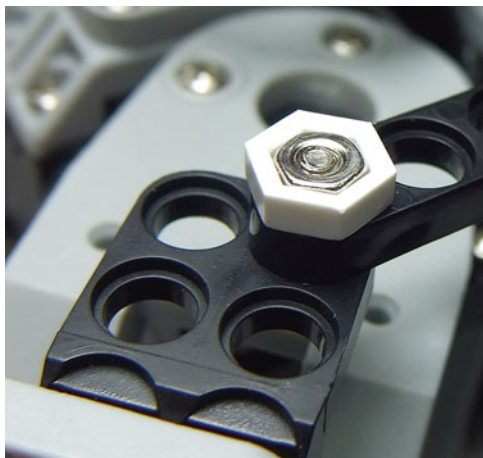
Fig. 2.10 CM-2+ (*left*—discontinued) and CM-700 (*right*) controllers

memory, but since 2013 it is shipped with the CM-530 which is based on the 72-MHz STM32F103RE with 256 KB flash memory. Otherwise, visually and functionally, the CM-510 and CM-530 are identical to each other. Both are also capable of handling embedded C applications via the WinAVR (CM-510) or WinARM (CM-530) tool chains (see further discussions in Chaps. 4 and 10) (Fig. 2.9).

It should be mentioned that there are also two barebone ATmega-based controllers, CM-2+ (discontinued) and CM-700, which were designed for custom needs when the user has to mix two types of Dynamixel modules together in the same controller (i.e., 3-pin TTL and 4-pin RS-485, see Chap. 3 for more details). Embedded C is also available for the CM-700 (Fig. 2.10).

The latest BIOLOID system (2012) is the STEM kit which combines hardware construction approaches from the previous BIOLOID kits (i.e., screws and nuts) and from the OLLO kits (i.e., plate and rivets). The STEM also has new hardware to create more secure pin joints (Fig. 2.11) and an IR Sensor Array (to be described later in Chap. 3). The STEM kit uses the CM-530 controller. It comes as two separate kits, Standard and Expansion, and the Standard kit is required for the proper use of the Expansion kit.

Fig. 2.11 BIOLOID
STEM pin-joint hardware



All BIOLOID systems use the RoboPlus software suite consisting of four tools:

1. **MANAGER**—for general hardware troubleshooting and obtaining firmware update for controllers.
2. **TASK**—general programming environment for the user.
3. **MOTION**—motion programming environment for multi-links robots.
4. **DYNAMIXEL WIZARD**—for troubleshooting and updating firmware on Dynamixel actuators and sensors.

The OpenCM platform is the vehicle for ROBOTIS to accomplish its open-hardware and software goals in the near future for operating systems such as MS Windows, Mac OSX and Linux. The CM-900 was the beta platform first available in 2012 but is no longer sold by ROBOTIS. It had two editions, ES and V.1.0, which are based on the STM32F103C8 microcontroller. They carried 64 KB of flash memory and support many hardware interface standards such as USB (1), CAN (1), USART (3), I2C (2) and SPI (2). The ES version supported both AX/MX-TTL and RS-485 Dynamixel ports, the V.1 edition additionally supported the new XL-TTL Dynamixel port. Both supported software development using an Arduino-based IDE (called “OpenCM IDE”) and wireless communications programming via ZigBee and Bluetooth. Real-time debugging (SWD & JTAG) was also available using additional tools such as ST-LINK/V2 and Keil μ Vision (Fig. 2.12).

Currently, only the OpenCM-9.04 series is commercially available and it comes in three versions A, B and C. The 9.04 series has the same hardware features previously listed for the 9.00 series, however they have a smaller physical format and 128 KB of flash memory (Fig. 2.13).

The B version is “ready-to-go” if the user plans to use a mixture of AX/MX-TTL and XL-TTL Dynamixel modules. The A version is essentially a user-customizable B version whereas the user can install only the needed headers. Both A and B versions are completely open hardware and software controllers whereas users can adapt their own firmware and boot loader as they wish. They use the OpenCM IDE as the programming interface.

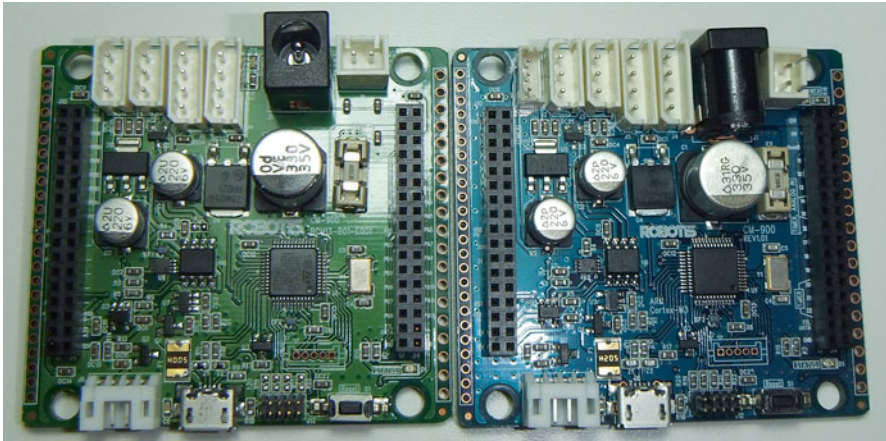


Fig. 2.12 CM-900 ES (left) and V. 1 (right)

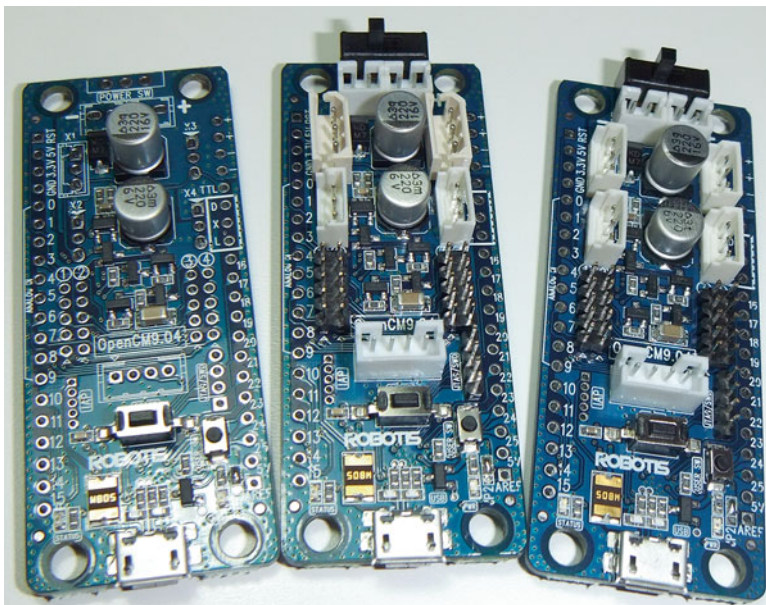


Fig. 2.13 OpenCM-9.04-A/B/C controllers (left to right)

Currently the C version comes with the ROBOTIS(DARWIN)-MINI kit and is also available by itself (<http://www.robotis.us/opencm9-04-c-with-onboard-xl-type-connectors/>). It has only the XL-TTL connectors installed. Most importantly, it has a proprietary firmware so that it can operate with the R+Task and R+Motion V.2 software suite. Alternately, the user can use the OpenCM IDE with version C but this mode would effectively erase the proprietary firmware, thus if the user

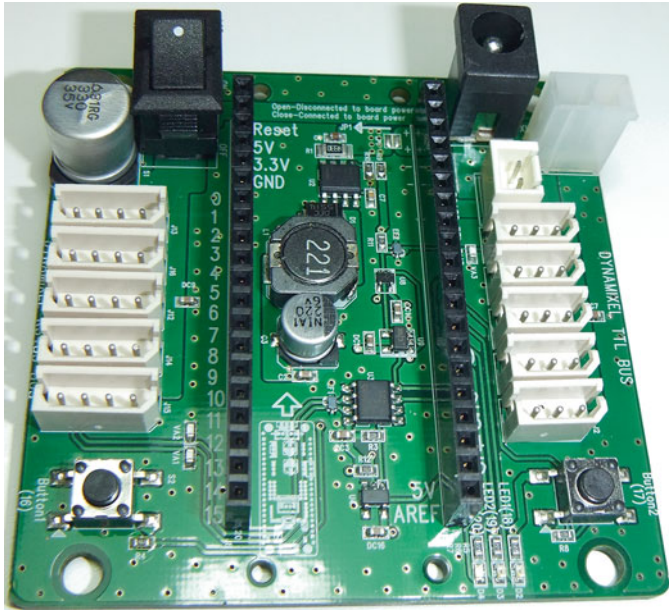


Fig. 2.14 Expansion board OpenCM-485-EXP for the OpenCM-9.04 series (beta version—*green*, release version—*blue*)

wants to use the R+Task and R+Motion packages again, a firmware recovery process must be performed (see Chap. 4).

If the user requires more TTL and RS-485 Dynamixel ports, an expansion board can be used (OpenCM-485-EXP) (Fig. 2.14).

2.3 Micom Training Kit

By the Summer of 2015, ROBOTIS plans to release a “minimal” training kit named “Micom Training Kit”. It is based on the OpenCM-9.04-C and oriented towards the DIY type of user. Figure 2.15 shows its main components:

- One OpenCM-9.04-C with 2-mm header mounted.
- Three NIR sensors (5-pin type).
- Two XL-320 actuators.
- One solderless breadboard and some 100 mm jumper wires.
- Some electrical and electronics components such as resistors (10 and 100 K Ω), one red LED, matching NIR LED emitter/receiver, one toggle switch, one variable resistor, a microphone and a 7-segment LED display.
- Additional OLLO frame and wheel parts to make a wheeled robot using NIR sensors so that it can follow a black track (for example).

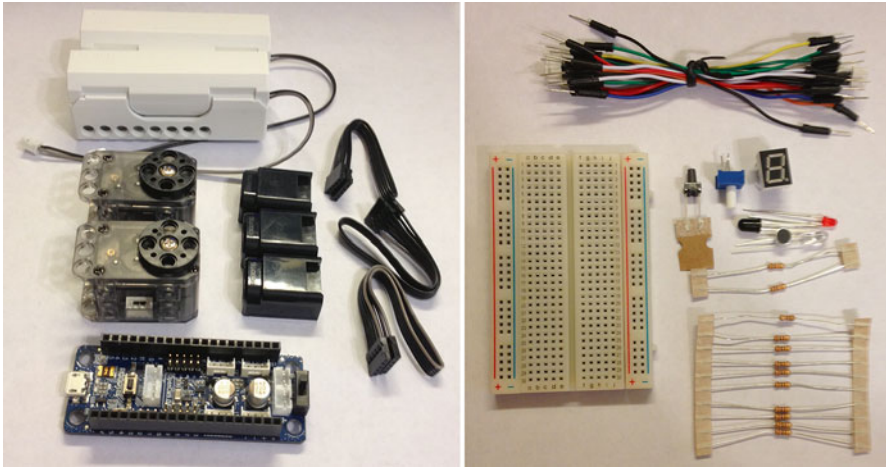


Fig. 2.15 Micom training kit (due Summer 2015)

With this kit, the DIY user can start programming using the TASK tool for a quick learning approach, then later can switch to the Arduino based ROBOTIS IDE for a low-level programming style using the C language and the breadboard for experimental circuits such as:

- Direct pin access and digital output (PWM and duty cycle).
- Analog input and A/D conversion.
- Serial communications.
- Dynamixel control and instruction packet management.
- Memory access, etc....

ROBOTIS has not finalized the price for the Micom Training Kit at this point in time (March 2015), but the author’s estimate is around \$150–175.

2.4 System(s) Selection Criteria

When I first prepared the instructional materials for my “Embedded Robotics” course in early 2009, the only option was the CM-5 based systems: BIOLOID COMPREHENSIVE and BIOLOID EXPERT. But nowadays, as shown in the previous sections, we have many more choices for entry points into this “Robotics” journey. Furthermore, ROBOTIS has been putting in diligent efforts for maintaining backwards compatibilities for their software updates and their newly developed actuators and sensors so that the CM-5 based systems are still useful. For example,

the following devices are compatible with the CM-5, although they may not function at their full capacities due to the 16 MHz clock speed on the Atmel AVR chip:

- IR Array Sensor (2012) (http://support.ROBOTIS.com/en/product/auxdevice/sensor/ir_sensor_array.htm)
- Servo motor MX-28T (2011) (http://support.ROBOTIS.com/en/product/dynamixel/mx_series/mx-28.htm)
- Color video camera HaViMo2 (2010) (https://www.havisys.com/?page_id=8)

So let me go out on a limb and share with you these recommendations:

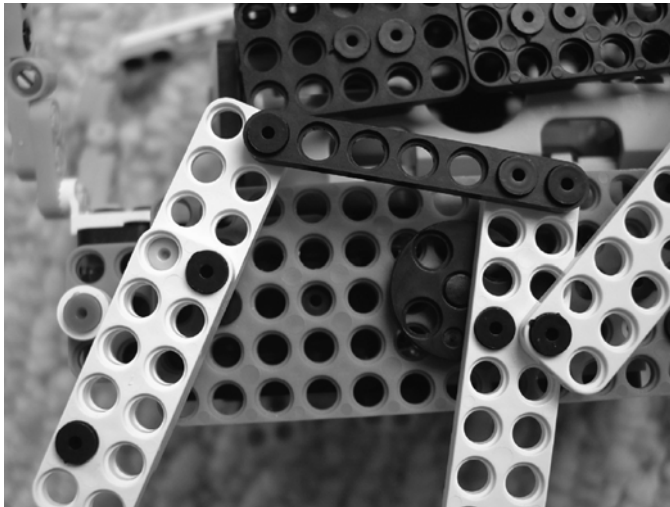
1. For the DIY user, the Micom Training Kit can be an economical approach whereas the user can start with TASK and progress to ROBOTIS IDE. This kit could turn out to be a popular entry kit for a first look at the ROBOTIS robotics ecology.
2. The CM-530 STEM or PREMIUM kits would offer a fast MCU @ 72 MHz and the 5-pin GPIO connectors, and Embedded C is readily available (via WinARM and Eclipse). If you have little background in computer programming, you can get started on RoboPlus Suite (Firmware 1.0) and transition to Embedded C with WinARM or purchase additional CM-9.04-A's or B's for an Arduino-style IDE, as all your hardware would still be compatible. ROBOTIS' e-Shop carries a Programming Guide showing how to use the RoboPlus V.1 software suite (http://www.robotis-shop-en.com/?act=shop_en.goods_view&GS=1486&GC=GD080400).
3. If you are already familiar with Arduino, there is good reason to use the CM-9.04-A/B from the beginning. But you will have to purchase all the other components separately as you need them. ROBOTIS still sells a frame-only kit at \$100 (http://www.robotis-shop-en.com/?act=shop_en.goods_view&GS=1269&GC=GD0803). The assembly manuals for all robots that can be built with the PREMIUM kit are available from the ROBOTIS e-Manual web site (http://support.ROBOTIS.com/en/product/bioloid/premiumkit/premiumkit_download.htm).
4. Option 4 is similar to Option 3, but it would use the ROBOTIS-MINI system instead. The CM-9.04-C is designed to work with the second generation of the RoboPlus software suite which has four tools: R+Design, R+Task, R+Motion and R+Manager. The first three tools are available since Summer 2014, but the fourth tool R+Manager probably would not be available to users until Summer/Fall 2015. This option would afford the users the most flexibility in switching between ROBOTIS-style and Arduino-style user interfaces.

For my part, I am committed to keep examples and projects described in this book compatible to systems from CM-5 to CM-9.04-C as much as possible.

2.5 Review Questions for Chap. 2

1. What is the product brand name uniquely attributed to ROBOTIS Co.?
2. What was the name of the first commercial product from ROBOTIS?

3. Which are the robotic system(s) from ROBOTIS that use a rivet-like system for building robots?
4. What are the micro-controllers currently supported by ROBOTIS?
5. List the robotic system(s) that are based on the STM32 family from STMicroelectronics Co.
6. List the robotic system(s) that are based on the ATmega AVR family from Atmel Co.
7. What is the name of the controller(s) associated with the open hardware-software initiative from ROBOTIS?
8. Which CM-XXX controller(s) can implement C-based computer programs from the user?
9. Which CM-XXX controller(s) run at 72 MHz?
10. List the four software tools that come with the RoboPlus Software Environment.
11. List three possible communication/interface protocols that are available to the user for use with ROBOTIS systems.
12. Which controller(s) support user-based firmware and boot loader?
13. Which controller(s) support an Arduino-based IDE?
14. Which controller(s) support an expansion board?
15. Name the four types of linkages that are found in a typical 4-bar linkage system.
16. A _____ linkage is called a _____ linkage when it can turn a full 360°.
17. **Identify** (i.e., draw) the correct links and **name** them properly for the 4-bar linkage system as shown for the Cricket robot's front leg picture below.



Chapter 3

Hardware Characteristics

In this chapter, the goal is to go over the main hardware characteristics of ROBOTIS' controllers, actuators and sensors so that the user can evaluate their hardware options in designing a robotic system to suit their needs.

Current ROBOTIS systems can be broadly divided into two groups, “Firmware 1.0” and “Firmware 2.0”. The microcontrollers used in “Firmware 1.0” systems are based on the Atmel AVR chip, except for the CM-530 which is based on an ARM Cortex M3 chip from STMicroelectronics. “Firmware 2.0” systems are all based on ARM Cortex M3 chips from STMicroelectronics (Figs. 3.1 and 3.2).

3.1 The Atmel AVR Family

3.1.1 CM-5 (*Discontinued*)

The CM-5 controller (c. 2005) is based on the ATmega128 clocked at 16 MHz and has 128 KB of flash memory. It comes with the BIOLOID BEGINNER and COMPREHENSIVE (BBC) kits.

It has three TTL-Dynamixel (3-pins) connections, thus ones can use:

1. AX and MX types of actuators, but only the AX-12A actuators come with the BBC kits.
2. Also available with the BBC kits is the AX-S1, an integrated sensor, with three NIR sensors (right, center, left) which can be used in both active and passive modes. It has a buzzer which can also serve as a sound clap counter. It also has an NIR receiver which can be used with the RC-100 remote controller in NIR mode.
3. Other Dynamixel-based sensors (3-pin) do exist for other types of measurements such as foot pressure, 3-axes acceleration or color video camera which will be discussed in more details in Sect. 3.3.

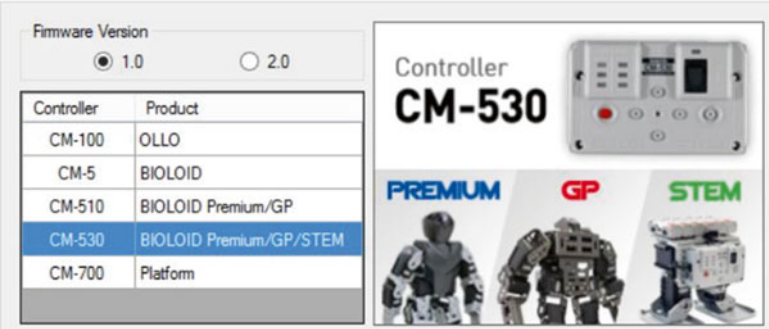


Fig. 3.1 “Firmware 1.0” systems

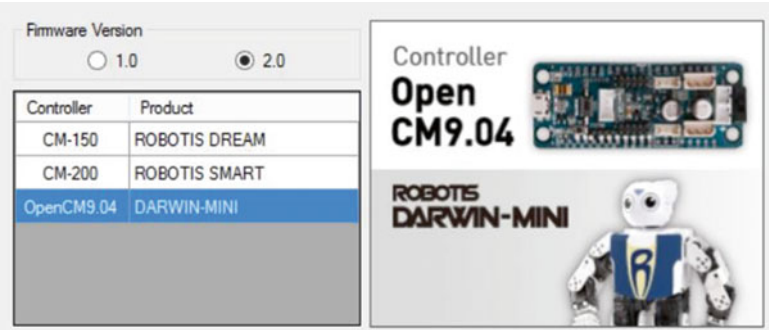


Fig. 3.2 “Firmware 2.0” systems

It uses a mini-jack connector/cable (BSC-10) to communicate with the PC via a 9-pin RS-232 connector. If your PC has only USB ports, you need to use the USB2Dynamixel module to serve as the interface between the USB port and the CM-5 communication port. More advanced users can also use the USB2Dynamixel to control 3-pin (TTL) and 4-pin (RS-485) types of actuators and sensors directly from the PC (C/C++ programming knowledge is required and several web sites have related information and tutorials such as “softwaresouls.com”, “forum.trosenrobotics.com” or “robosavvy.com/forum”).

The CM-5 is also capable of wireless communications via the ZIG-100 daughter card. However it can only accommodate only ONE mode of communication at any one time, i.e., either wired or wireless but not both. With the “wired” option, ones can download microcontroller codes as well as send “remote control” commands to the CM-5 via the Virtual RC-100 Controller. The “wireless” option only allows remote control commands via a physical RC-100 Controller (equipped with a matching ZIG-100) or via another CM-5 (also with a matching ZIG-100). Please note that since 2012, the RC-100 got discontinued and was replaced by the RC-100A/B to accommodate additional BlueTooth communications features

(<http://www.robotis.us/rc-100b/>). More detailed ZigBee and BlueTooth communications programming will be discussed in Sect. 5.6 and Chap. 8 (Fig. 3.4).

If your PC has a 9-pin RS-232 connector (assumed to be COM1), you can adopt the following scheme to use the “wired” connection for programming and debugging, and then switch to “wireless” for operating your robot during run-time:

1. Using the black 9-PIN to mini-jack serial cable hooked up to COM1, keep on using the TASK tool via COM1 to create your TSK program and download it to the CM-5 controller and use the Virtual Controller RC-100 as normal to check out your code (Fig. 3.5).
2. Use a spare USB port to plug in a combination module (USB2Dynamixel-Zig2Serial-ZIG-100, see Fig. 3.6)—in my particular case, it turned out to be COM34. Next, UNPLUG the mini-jack serial cable FROM the CM-5 (if you leave this mini-jack connector plugged in, the CM-5 disables its ZigBee circuit) (Fig. 3.6).

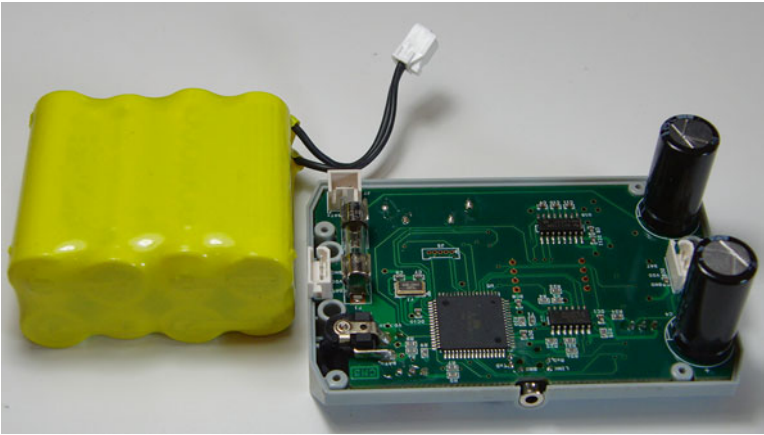


Fig. 3.3 Internal view of the CM-5

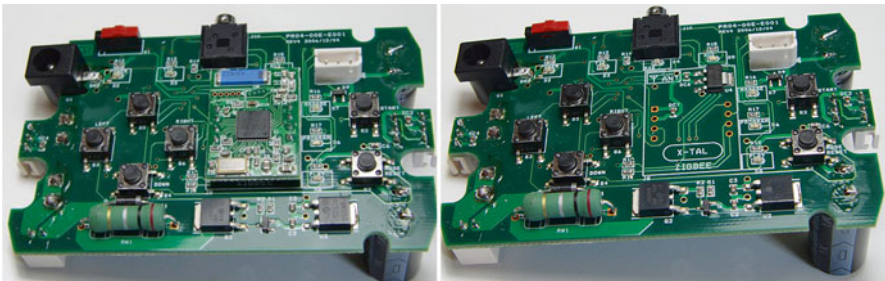


Fig. 3.4 CM-5 with (*left*) and without (*right*) daughter card ZIG-100

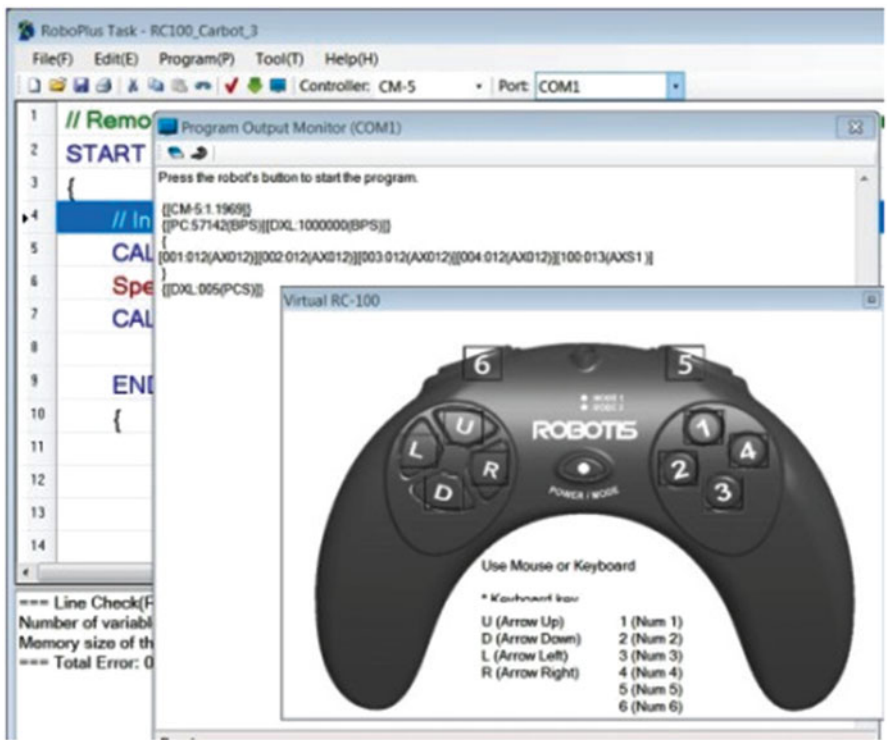


Fig. 3.5 Using CM-5 via wired connection (COM1)

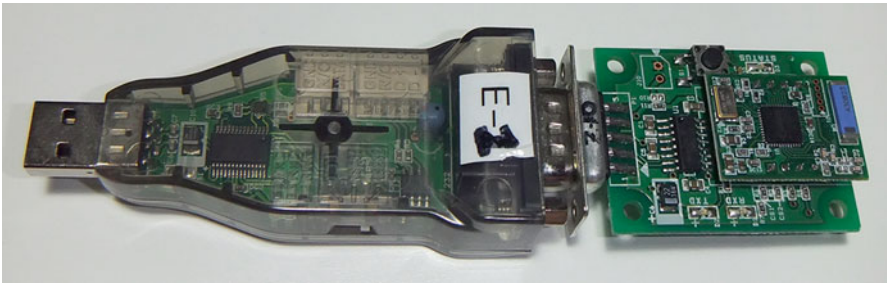


Fig. 3.6 USB2Dynamixel-Zig2Serial-ZIG-100 combo

3. Make sure that you switch to COM34 in TASK tool, then click on “View Output of Program” on the menu bar to open up the Output Window (see Fig. 3.7) and then use the Virtual RC-100 as normal, except now it is really routed wirelessly via COM34—also make sure that you actually have the CM-5 in play mode and had started the downloaded program as normal.

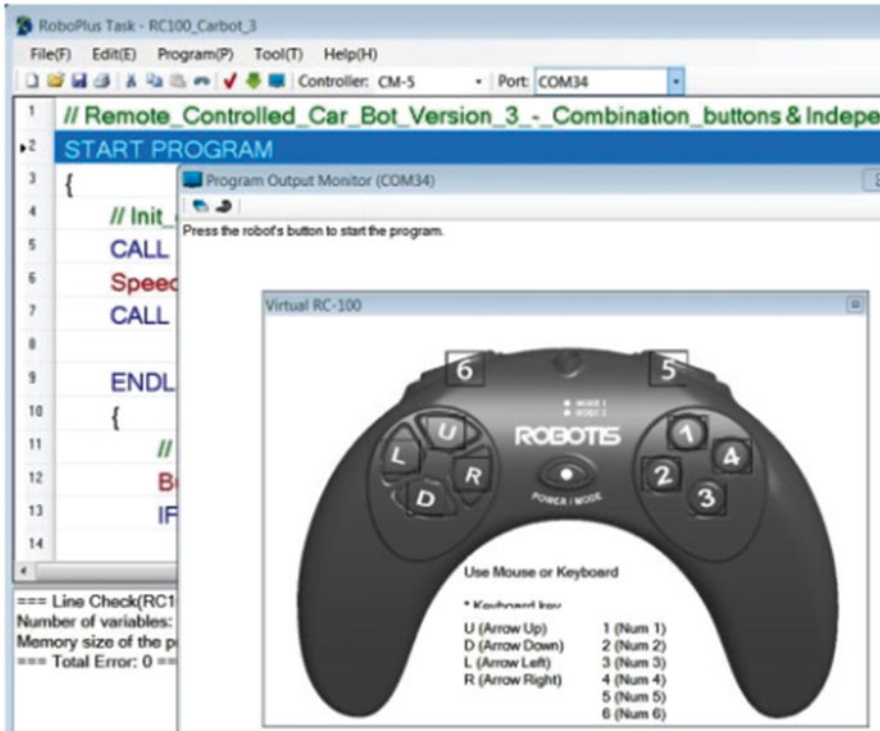


Fig. 3.7 Using CM-5 via wireless connection (COM34)

3.1.2 CM-510 (Discontinued)

The CM-510 controller (c. 2009) is based on the ATmega256, still clocked at 16 MHz but has 256 KB of flash memory (Fig. 3.8). It comes with the BIOLOID PREMIUM (BP) kit from 2009 to 2012. After 2012, the CM-510 is replaced by the CM-530 for the BP kit.

It has an extended suite of Dynamixel, I/O and communications ports:

1. Five 3-pin TTL Dynamixel ports (work with AX and MX actuators and other Dynamixel based sensors).
2. Six 5-pin GPIO ports (work with sensors such as NIR (intensity-based, distance measurement (NIR reflected-angle-based), gyroscope, touch, color, magnetic, with temperature, ultrasonic and motion recognition sensors coming up in 2015).
3. One 4-pin communication port for the ZIG-110A which is the equivalent of the ZIG-100 (Fig. 3.9). All previously discussed PC communication options for the CM-5 still apply for the CM-510 (Sect. 3.1.1).

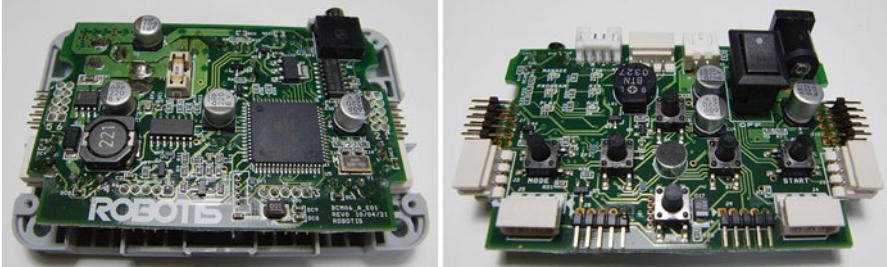


Fig. 3.8 Internal views of the CM-510

Fig. 3.9 ZIG-100 and ZIG-110A ZigBee communication modules

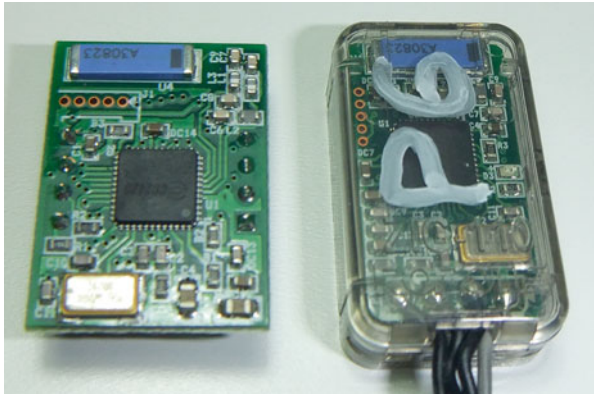


Fig. 3.10 BlueTooth communication devices BT-100/110A and BT-210

- Starting in 2012, BlueTooth communications devices were added: BT-100/110A and BT-210 (Fig. 3.10). They can be used with the same 4-pin communication port on the CM-510 but with different features unique to the BlueTooth protocol. The BT-100/A will only work in the RC-100A/B remote controller. When using BT modules, the user must take note that the PC would associate two COM ports

to each BT module used, one OUT-GOING and one IN-COMING (the user can check this information via Windows Device Manager). Thus when using ROBOTIS' application software on the PC side, the user needs to make sure to choose the OUT-GOING COM port.

5. The CM-510 has a sound generator and detector built in its circuit board, and still uses the serial mini-jack cable to communicate with the PC for the “wired” option.

3.2 The STM ARM Cortex M3 Family

3.2.1 CM-530

The CM-530 controller (c. 2012) is based on the STM32F103RE, clocked at 72 MHz and has 256 KB of flash memory (Fig. 3.11). It uses the USB port directly (i.e., no need for the USB2Dynamixel module). Otherwise it is functionally and visually identical to the CM-510.

It has the same extended suite of Dynamixel, I/O and communications ports as the CM-510:

1. Five 3-pin TTL Dynamixel ports.
2. Six 5-pin GPIO ports.
3. One 4-pin communications port, and one USB-mini port.

All actuators, sensors, ZigBee and BlueTooth modules previously discussed for the CM-510 (Sect. 3.1.2) work with the CM-530 with a boost in MCU clock rate (72 MHz instead of 16 MHz).

Since Spring 2014, there had been an issue on the PC side with the FTDI driver for the USB port, a driver reinstallation would be needed and a guide is available at http://www.robotis.com/xe/download_en/646927.

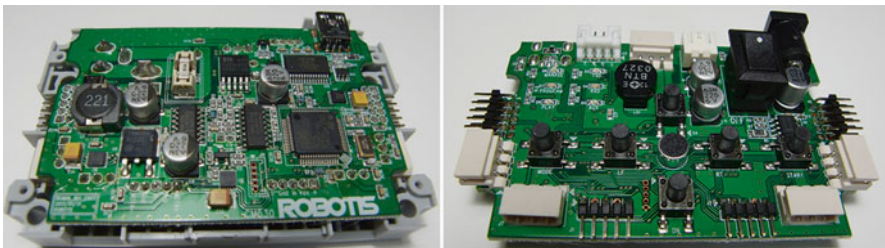


Fig. 3.11 Internal views of the CM-530

3.2.2 CM-900 (Discontinued)

The CM-900 series was ROBOTIS’ entry into the open hardware and software movement and the CM-900 controller was available to the beta testing community between October 2012 and July 2013. It had two versions, ES and V.1 (see Fig. 2.12) and both were clocked at 72 MHz. V.1 was only commercially available during 2013.

The main technical specifications for the CM-900 V.1 are listed below:

	CM-900 V.1
CPU	STM32F103C8 (ARM Cortex-M3)
Op voltage	5–35 V (USB 5 V, DXL 12 V, 7.4 V)
Flash	64 Kb
3-Pin DXL TTL	2
4-Pin DXL RS485	2
Mini 3-Pin DXL TTL	1

From the DXL connector and operating voltage options shown above, ones can see that the CM-900 was quite possibly envisioned to control the complete family of Dynamixel actuators from AX to Dynamixel Pro. It used an Arduino-based IDE called OpenCM IDE (Fig. 3.12).

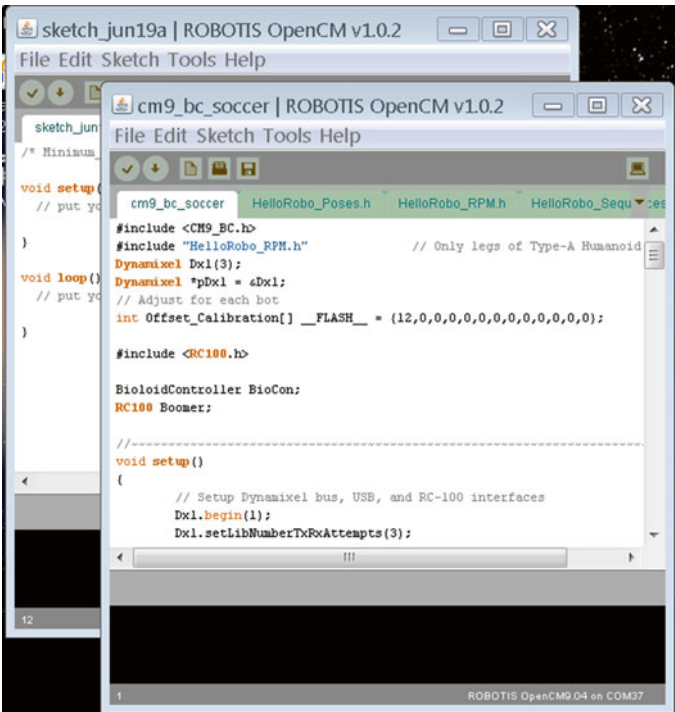


Fig. 3.12 OpenCM IDE V.1.0.2

Included with this book is a zipped file containing user manuals and other technical information for the CM-900 series (OpenCM-900.zip). These files originated from the CM-9 Developer’s World Circle at http://www.robotsource.org/bs/bd.php?bt=forum_CM9DeveloperWorld where interested users can get more updated information.

With feedbacks from the beta test community, the CM-900 was redesigned into the OpenCM-9.04 in July 2013.

3.2.3 OpenCM-9.04 Series

The OpenCM-9.04 series illustrates a new segmented-market view by ROBOTIS which currently offers three versions of the OpenCM-9.04 (A, B and C) with the same main technical features listed below:

	CM-9.04-A/B/C
CPU	STM32F103CB (ARM Cortex-M3)
Op voltage	5–16 V (USB 5 V, DXL 12 V, 7.4 V)
Flash	128 Kb

The OpenCM-9.04-A (Fig. 3.13) allows the user complete freedom in mounting only the components that are needed for his or her project from the power switch to the types and numbers of interface headers needed for AX/MX-TTL or XL-TTL (3-pin) and GPIO (5-pin), and even JTAG/SWD connection.

The OpenCM-9.04-B (Fig. 3.14) comes pre-connected for power switch, battery connections and connectors for AX/MX-TTL (2), XL-TTL (2), GPIO (4), JTAG/SWD (4-pin) and wired/wireless communications (4-pin).

Fig. 3.13 OpenCM-9.04-A controller

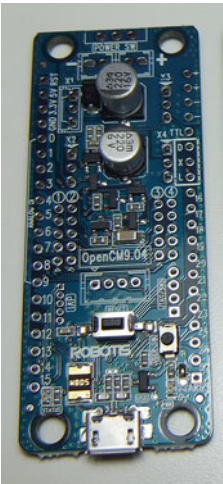


Fig. 3.14 OpenCM-9.04-B controller



Fig. 3.15 OpenCM-9.04-C controller



The A and B versions have open firmware and are designed to work only with the ROBOTIS IDE software. Included with this book is a zipped file containing user manuals and other technical information for the OpenCM-9.04-A/B series (OpenCM-904.zip) which originated from the ROBOTIS e-Manual web site <http://support.robotis.com/en/product/controller/opencm9.04.htm> where interested users can get more updated information. Advanced users may also want to invest in the in-circuit debugger and programmer ST-LINK/V2 (<http://www.st.com/web/en/catalog/tools/PF251168>) and Keil μ Vision IDE (<http://www.keil.com/arm/mdk.asp>) to modify firmware and bootloader programs.

The OpenCM-9.04-C (Fig. 3.15) comes pre-connected for power switch, battery connections and connectors for Mini-DXL-TTL (4), GPIO (4) and JTAG/SWD (4-pin) and wired/wireless communications (4-pin).

The C version comes with a proprietary firmware to make it function with the next generation of ROBOTIS software: R+Task, R+Design, R+Motion and R+Manager. Alternately, the user can use the OpenCM IDE with version C but this mode would effectively erase the proprietary firmware, thus if the user wants to use the R+ packages again, a firmware recovery process must be performed.

If the user requires more AX/MX/XL-TTL and/or RS-485 Dynamixel ports (i.e., additional power options), an expansion board can be used (OpenCM-485-EXP) and the user also needs to solder extra headers (from an accessory set) onto the OpenCM-9.04, please visit the web links below for more information:

- http://support.robotis.com/en/product/controller/opencm_485_exp.htm
- http://www.robotis-shop-en.com/?act=shop_en.goods_view&GS=1623&GC=GD080201
- http://www.robotis-shop-en.com/?act=shop_en.goods_view&GS=1490&GC=GD080201

As compared to the CM-5XX series, the OpenCM-9.04 series have more options regarding connections from the PC: plain USB cable (A to micro), or LN-101, ZigBee/Bluetooth via Serial 2 (USART CH 2) (see Fig. 3.16). Serial 1 (USART CH 1) was still reserved to the Dynamixels on the local OpenCM-904 Dynamixel TTL bus. Serial 3 was “new” and connected to the OpenCM 485 Expansion Board for access to AX/MX/DX/RX/PRO types of Dynamixel actuators.

Fortunately, a compatibility chart for all ROBOTIS products can be found at http://support.robotis.com/en/product/controller_main.htm, however a word of caution when using that list, for example:

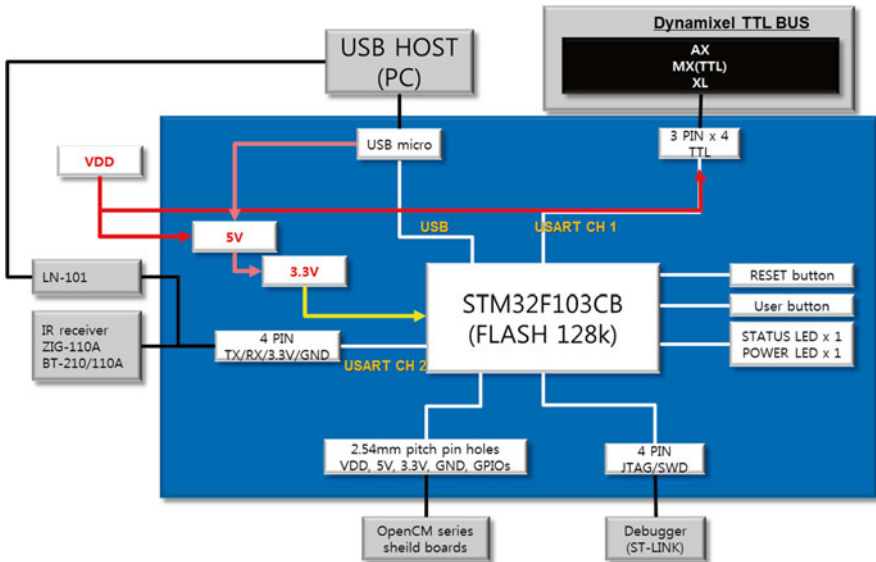


Fig. 3.16 Functional block diagram for OpenCM-9.04 series

- A plain USB connection would work best for an OpenCM-9.04-A/B controller if you use the ROBOTIS IDE (Arduino-based), but it won't work if you try to use it with the RoboPlus Manager tool to update its firmware. However, the user can update the OpenCM-9.04-C firmware via this USB cable and RoboPlus Manager (see more details in Sect. 4.2.2). A more versatile module is the LN-101 that would allow the Dynamixel Wizard tool to update firmware (at the same time) on a CM-9.04-C and the XL-320 s connected to it (see Sect. 4.1.2.1). However, the LN-101 won't connect to the OpenCM-9.04-C controller to show the regular Manager's interface with all the hardware settings for this controller (hopefully this situation will be resolved in a future version of Manager).
- The ZigBee (ZIG-110A) and Bluetooth modules (BT-110A and BT-210) are compatible with the 9.04-C, but so far I had found that ZigBee performed better than Bluetooth in terms of connection latency and packet performance (more on this in Chap. 4).

Ironically, part of these issues is due to the fact that ROBOTIS is committed to continuing development of their hardware and software systems, so the software team is always playing catch-up to the hardware team! In Chap. 5, "particular" usages will be described in more details.

Furthermore, starting in Summer 2015, ROBOTIS plans to release a new series of Bluetooth modules (BT-410) based on Bluetooth 4.0 Low Energy (BLE) standard. This series will be able to handle 1 to 1 (1:1) and 1 to N (1:N) communications and targeted towards mobile devices (see Fig. 3.17).

Fig. 3.17 BT-410 series
(available Summer 2015)



3.3 The Dynamixel Actuators Family

ROBOTIS has an extensive repertoire of actuators and maintains up-to-date technical information at this web link (http://support.robotis.com/en/product/dxl_main.htm). Currently most of ROBOTIS actuators are using Communication Protocol 1 (http://support.robotis.com/en/product/dynamixel/dxl_communication.htm), except for the XL-320 and Dynamixel PRO which are using Communication Protocol 2 (http://support.robotis.com/en/product/dynamixel_pro/communication.htm). The goal of this section was not to provide duplication of the above information but to provide the reader with some complementary hardware information using the AX-12 and MX-28 as examples.

The MX-28 was chosen as a representative of the types that can perform 0–360° in position control mode and with PID control features, while the AX-12 was chosen as a representative of the types that have a more limited range of motion from 0 to 300° when in position control mode and with only proportional control.

3.3.1 *The AX Series*

The first ROBOTIS Dynamixel actuator was the AX-12 (c. 2003) which had a stall torque of 1.5 Nm and ran around 59 RPM. Throughout the years, it got updated to AX-12+ and then to AX-12A (current version). It has two “cousins”:

1. AX-18F/AX-18A which has a slightly higher stall torque of 1.8 Nm and runs faster at 97 RPM.
2. AX-12W which is designed more for continuous rotation as wheels. It can rotate in wheel mode at 470 RPM and in joint mode at 54 RPM. Its stall torque is not provided by ROBOTIS.

They all use DC motors and have proprietary firmware implemented on an Atmel AVR microcontroller (16 MHz and 8 KB memory). Their firmware can be updated via the Dynamixel Wizard software tool, using a USB2Dynamixel module and requiring separate power (via the SMPS2Dynamixel, for example). See this web link for more details (http://support.robotis.com/en/software/roboplus/dynamixel_wizard.htm). In operations, the actuators get updated every 8 ms by the main controller via the Dynamixel bus. The TTL communication speed for the AX family can be set between 7.8 Kbps and 1 Mbps.

The good folks at Irish Robotics have provided circuit diagrams for the AX-12 motor and processor, and even for the CM-5 at the following links:

<http://robosavvy.com/Builders/Chris.H/AX12Motor.pdf>
<http://robosavvy.com/Builders/Chris.H/AX12Processor.pdf>
<http://robosavvy.com/Builders/Chris.H/CM5.pdf>

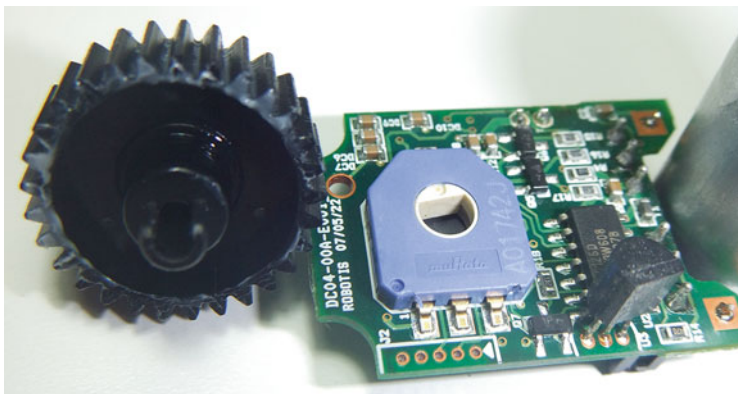


Fig. 3.18 AX-12 PCB with Murata SV01A103 (*light gray hexagonal box*) and main gear shaft

One AX-12 feature that users found most intriguing was how could the AX-12 rotate completely in 360° when in wheel mode but was restricted to a 300° range when in position control mode. This was due to the position encoder used on the AX-12 which is a Murata SV01A103, effectively a potentiometer with an effective range of only about 333° (see Fig. 3.18).

During assembly at the factory, the main gear shaft (black plastic) is inserted into the semi-circular white rotor of the Murata part and then through a gear train it is connected to the DC motor (silver cylinder on the right in Fig. 3.18). Thus when the motor rotates, the white rotor also rotates but at a slower speed due to the gear ratio. The white rotor “wipes” against a resistive element inside the SV01A103 and referring back to the AX12Motor.pdf document, ones can see that when at a position 60° left of the “12 o’clock” position, this rotor (Pin 2) would provide a 5 V reading to the AX-12 microcontroller which would then digitize it to a numerical value of 1023. Similarly, when this rotor reaches 60° left of the “6 o’clock” position, Pin 2 would provide a 0 V reading which gets digitized to a numerical value of 0. And thus there is a $\pm 30^\circ$ zone around the “9 o’clock” position where Pin 2 is not connected to the resistive element, and therefore no voltage reading is possible, and consequently the AX-12 microcontroller has no valid digitized value also (i.e., it does not know where it is at). This is why on the horn of the AX-12, there is an “I” mark (see Fig. 3.19) so that the user can use it to put the servo into the vicinity of the “512” position (3 o’clock position) in the assembly instructions for most robots, to be used as an initial pose before they perform any programmed action.

Ones can also recognize that with wears due to usage, the rotor may provide “unreliable” voltage readings to the AX-12 controller at certain positions and as a consequence of this, ones can observe that an older servo tends to “tremble” when it is commanded to stay at those “unreliable” positions.

Fig. 3.19 “T” mark on horn of AX-12 actuator

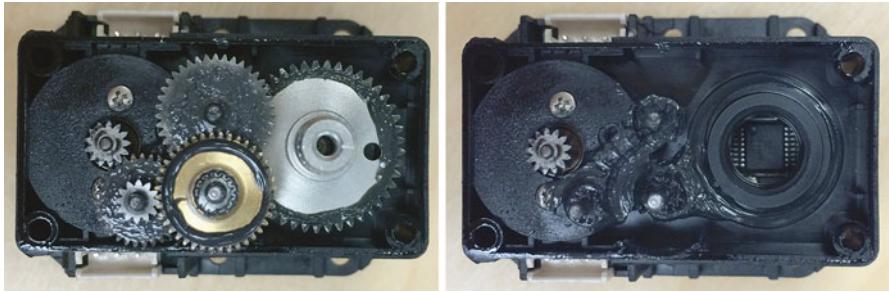
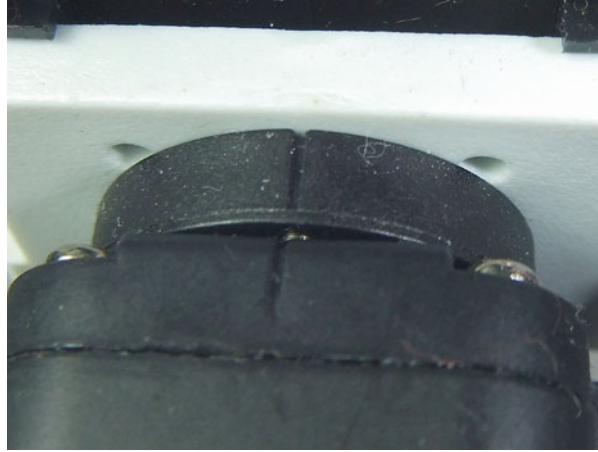


Fig. 3.20 (a, b) Internal views of MX-28T

As plastic gears can break off sometimes, users can order replacement sets at http://www.robotis-shop-en.com/?act=shop_en.goods_view&GS=1397&GC=GD080307. DIY videos on YouTube described the detailed steps plus tools needed to perform these maintenance tasks (<http://www.youtube.com/watch?v=W1sOavdmIus>).

Thus on higher-end actuators such as the MX-28, metallic gears and shafts are selected and most importantly contact-less magnetic position encoders are used.

3.3.2 The MX Series

The MX series started with the MX-28 which was designed for the ROBOTIS-OP (c. 2011). It has a stall torque of 2.5 Nm at 12 V and 55 RPM. This series use the STM32F103C8 @ 72 MHz as controllers and they has all metallic gears (see Fig. 3.20a). Once the gear train is removed, ones can see the magnetic encoder chip AS5045 inside the shaft cavity (Fig. 3.20b).

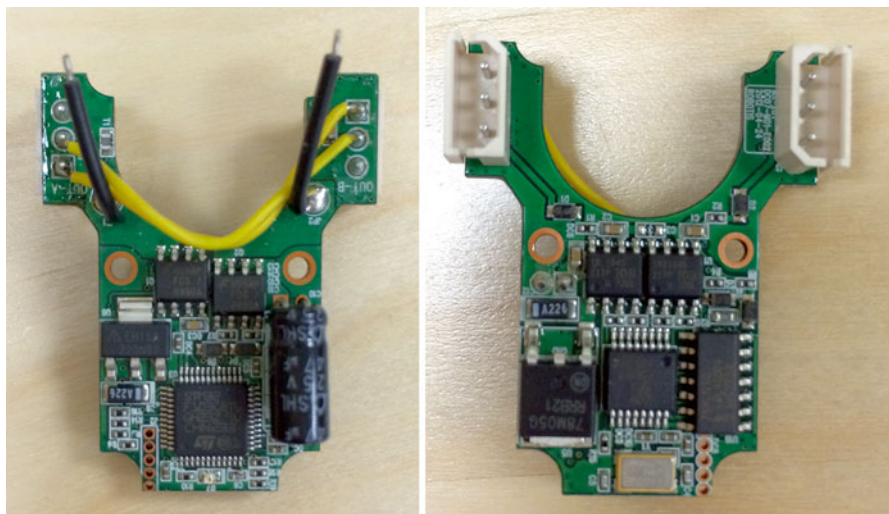


Fig. 3.21 (a–b) Two views of the PCB of an MX-28T

On the right in Fig. 3.20a, the big silver gear (where the servo's horn would be mounted to) has on the other end of its shaft a 2-pole magnet that would hover over the encoder chip AS5045 shown in Fig. 3.20b (see Fig. 17 on page 24 of the data sheet available at <http://www.ams.com/eng/Products/Position-Sensors/Magnetic-Rotary-Position-Sensors/AS5045>). When the motor turns, this magnet also turns via the gear train, and thus changes the magnetic field characteristics as sensed by the AS5045 which uses its own firmware to output a corresponding PWM signal over a full turn of 360° .

Figure 3.21a, b shows two views of the PCB of a typical MX-28T. In Fig. 3.21a one can see the main STM controller, and on Fig. 3.21b the magnetic encoder AS5045 is shown. This magnetic encoder allows the MX actuators to have position control for the complete $0\text{--}360^\circ$ range at 0.088° resolution (i.e., 12 bits).

Information for the complete MX family can be found at this web link: http://support.robotis.com/en/product/dynamixel/dxl_mx_main.htm. The TTL communication speed for the MX family can be set between 7.8 Kbps and 4.5 Mbps. The MX-28 has three “cousins”:

1. MX-12W, designed to be used as a wheel running at 470 RPM.
2. MX-64, with stall torque at 6.0 Nm and 63 RPM.
3. MX-106, with stall torque at 8.4 Nm and 45 RPM.

The effects of PID control features (MX-28) or the lack thereof (AX-12) on the position control behavior of these example actuators will be explored in more details in Chap. 6.

For a more general understanding of servo motor design and control, I would like to refer the readers to Clark and Owings (2003) or Kanniah et al. (2014). The “Humanoid Robot Lab” in Portugal had done detailed studies on the Bioloid Comprehensive kit and AX-12 (<http://humanoids.dem.ist.utl.pt/humanoid/overview.html>). And there are many more Internet sites sharing others’ knowledge and experiences with ROBOTIS products. ROBOTIS also maintains a ROBOTIS channel on YouTube showing many how-to videos for various maintenance tasks (<http://www.youtube.com/user/ROBOTISCHANNEL/videos>).

3.4 ROBOTIS Sensors Family

Currently available sensors from ROBOTIS and third-party partners can be sorted into two groups:

1. Dynamixel-compliant sensors (3-pin TTL), i.e., they can be daisy-chained on the Dynamixel bus and individually assigned with a unique ID (0–253, however ID 200 is reserved for the main microcontroller), for example:
 - (a) AX-S1, Integrated Sensor from ROBOTIS, NIR intensity-based sensor (short and long range active modes or passive mode), sound claps detector and NIR remote control sensor (http://support.robotis.com/en/product/auxdevice/sensor/dxl_ax_s1.htm).
 - (b) IRSA, IR Sensor Array from ROBOTIS, seven NIR intensity-based sensors with buzzer features (http://support.robotis.com/en/product/auxdevice/sensor/ir_sensor_array.htm).
 - (c) AX-S20, first Inertia Measuring Unit from ROBOTIS, 3-D acceleration components and resulting XYZ rotational angles, plus a magnetic heading sensor. In beta tests in 2010, but did not get to full production.
 - (d) Foot Pressure Sensor (from HUV Robotics) (http://www.huvrobotics.com/shop/index.php?_a=viewProd&productId=4). Not commercially available currently.
 - (e) HaViMo2, color video camera with onboard processing capabilities (https://www.havisys.com/?page_id=8). Commercially available in Europe, Mexico and Southeast Asia.
2. GPIO-based sensors (5-pin). These sensors have a open architecture and are attached to specific ports available on various CM controllers:
 - (a) CM-510 and CM 530 have six GPIO ports, while CM-900 and OpenCM9.04 have only four GPIO ports.
 - (b) From ROBOTIS, we have a growing supply of these types of sensors (IR, Distance Measurement DMS, Gyroscope, Touch, Color, Magnetic) and in the near future (Temperature, Ultrasonic and Gesture Recognition).
 - (c) The reader can find other third party sensors interfaced to the OpenCM system at http://www.robotsource.org/bs/bd.php?bt=forum_CM9DeveloperWorld.

3.4.1 AX-S1 and IRSA

Both sensors work on the principle of sending out 38-KHz pulses of NIR light (invisible to the human eyes) and then to process the NIR intensities reflected off objects in its path. Thus an object closer to the sensor would reflect more NIR energy (represented as a bigger numerical value to the microcontroller) than an object further away. In Fig. 3.22, the “clear” LED (right LED) sends out the NIR pulses, and the “black” LED (left LED) receives the reflected NIR intensities. Please note the “recessed” mounting of these LEDs to reduce interferences from each other.

This type of sensor does have a drawback when your robot has to deal with objects with different brightnesses as darker objects will be “perceived” in being “further away” than lighter objects although their “true” physical distances may be the same. Furthermore their response is non-linear, including a reversal in response at extreme close range (see Fig. 3.23).

However this technique is quite effective in detecting white from black areas on a complicated track using an IR Sensor Array as shown in Fig. 3.24.

The AX-S1 can switch between a short range detection (up to 1–3 cm) and a long range detection (up to 10–12 cm) using a software parameter. It also has a passive mode to detect light sources. The AX-S1 can use its microphone to sample at 3.8 KHz sound events such as hand claps, but they have to be at least 80 ms apart.

Fig. 3.22 Forward looking set of NIR LEDs for AX-S1

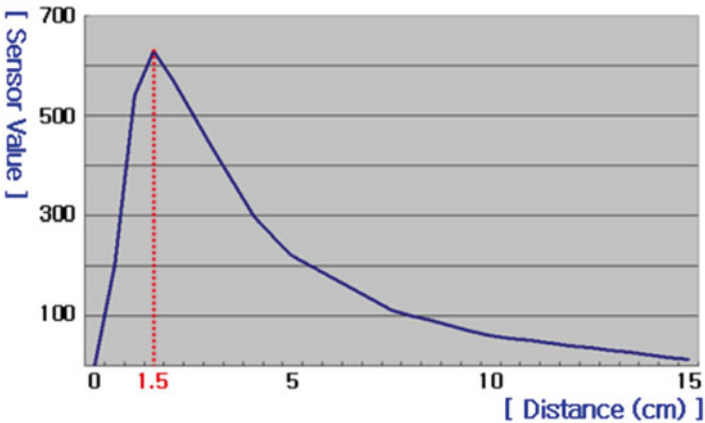


Fig. 3.23 Non-linear response of a typical NIR-LED sensor vs. distance

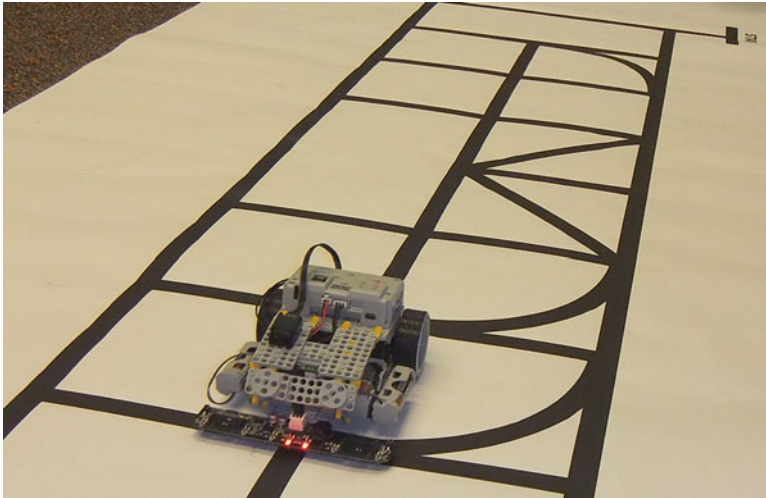


Fig. 3.24 Application of IRSA on a challenging track

These Dynamixel compliant sensors are also well documented at these ROBOTIS web links:

- http://support.robotis.com/en/product/auxdevice/sensor/dxl_ax_s1.htm.
- http://support.robotis.com/en/product/auxdevice/sensor/ir_sensor_array.htm.

More detailed applications programming of these sensors will be shown in Chap. 5.

3.4.2 AX-S20 (Discontinued)

Although this product never saw full production, the reader may be interested in its working. The AX-S20 used an ATmega8 for controller (i.e., Dynamixel compliant) and the sensor AMS0805WAH from Amosense (<http://www.motionsense.net/kor/index/AMS0805WAH%20DataSheet%201.3%20%28AMOSENSE%29.pdf>) for measuring 3-Axis Magnetic and Acceleration fields. The AMS0805WAH has an embedded calibration algorithm which eliminates the need for initial calibrations and supports precise calculation of motion data under all environments. It has 1° resolution for its Roll, Pitch and Azimuth angles. In practice, it supports about a 20 Hz sampling rate on a CM-510. Figure 3.25 shows an AX-S20 installed in the head piece of a PREMIUM Humanoid A robot and the resulting coordinates system (as a side note, the AX-S20, with its magnetometer, was mounted in the robot's head so as to keep it away from the electromagnetic interferences of the operating actuators as much as possible).

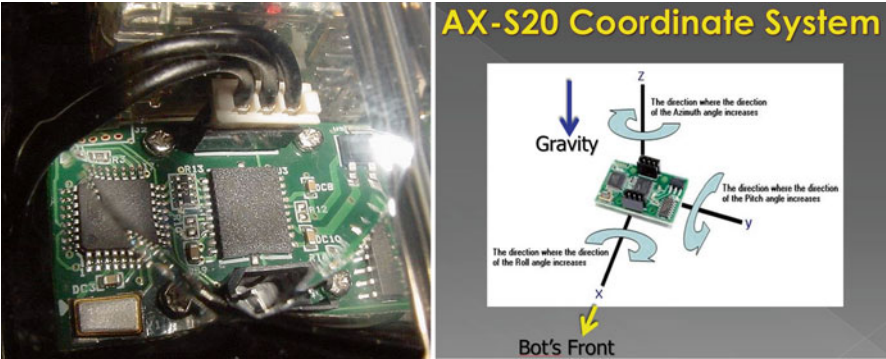


Fig. 3.25 AX-S20 installed in the head of a PREMIUM Humanoid A robot and the resulting coordinates system

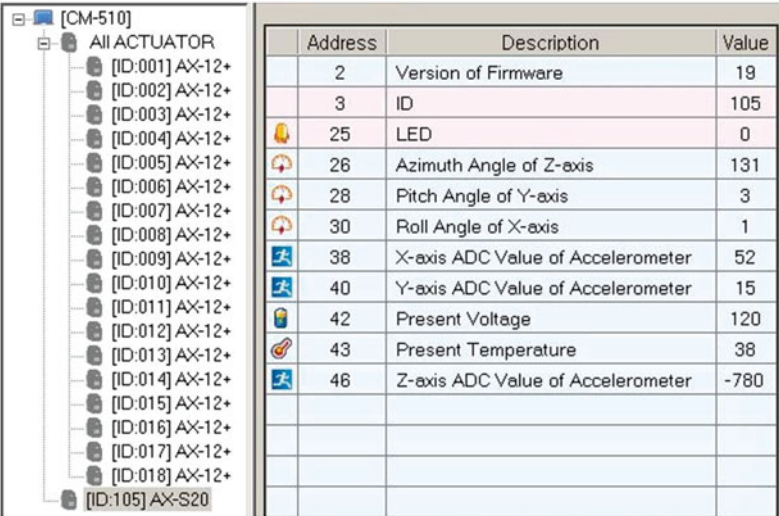


Fig. 3.26 Screen capture of RoboPlus Manager used on a PREMIUM Humanoid A robot with AX-S20 (ID=105)

Figure 3.26 is a screen capture of a RoboPlus Manager session on a Humanoid A robot with an AX-S20 (ID=105), and ones can see the list of various inertia parameters that ones can use in its programming.

So far we only got the opportunity to test the AX-S20 via the RoboPlus software suite and these application programming projects are discussed later in Chap. 9.

A newer IMU (MPU-9150) with gyroscope, accelerometer and compass components (each having three axes) is available at Spark-Fun Electronics

(<https://www.sparkfun.com/products/11486>) and has been adapted to work with the OpenCM-9.04 framework (http://www.robotsource.org/bs/bd.php?bt=forum_CM9DeveloperWorld&bt_id=707).

3.4.3 Foot Pressure Sensor (FPS: From HUV Robotics)

This HUV Robotics product is also Dynamixel compliant, but it is not currently available commercially (http://www.huvrobotics.com/shop/index.php?_a=viewProd&productId=4). Figure 3.27 shows how a set of FPS was mounted at the four corners of a foot of a PREMIUM Biped.

Each pressure sensor relies on a material that would change its electrical resistance (i.e., its voltage values) depending on the pressure imposed on it. These FPS require a flat hard surface for a satisfactory performance. Figure 3.28 shows the addressing scheme to access these values from inside a typical TASK program.

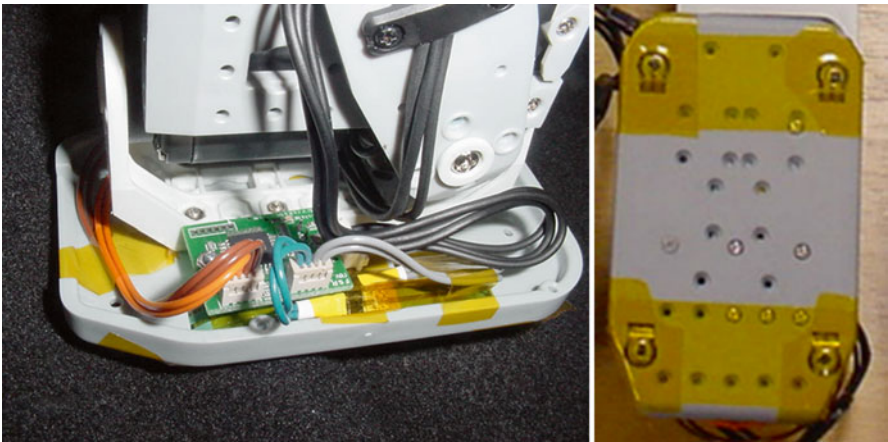


Fig. 3.27 HUV FPS mounted on foot of a PREMIUM Biped robot

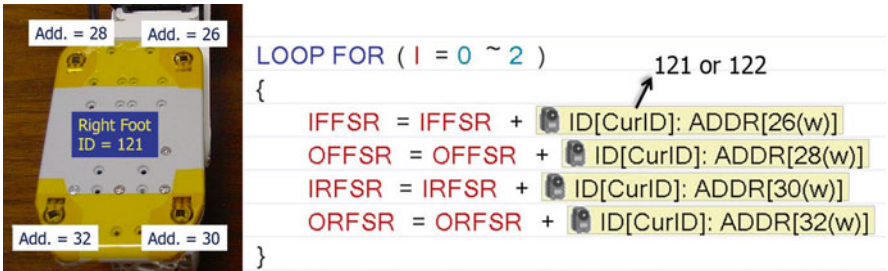


Fig. 3.28 Addressing scheme to access FPS data from a TASK program



Fig. 3.29 HaViMo 2.0 video cameras mounted on PREMIUM BiPed

So far we only got the opportunity to test the HUV FPS via the RoboPlus software suite and these application programming projects are described later in Chap. 9.

ROBOTIS provides a similar product to the HUV FPS but only for the ROBOTIS-OP (DARwIn-OP) called FSR (http://www.robotis-shop-en.com/?act=shop_en.goods_view&GS=1442&keyword=FSR).

3.4.4 *HaViMo 2.0*

This color video camera (Fig. 3.29) is available via e-shops located in the United Kingdom and Southeast Asia (https://www.havisys.com/?page_id=8).

The HaViMo 2.0 is Dynamixel compliant and it has a video frame resolution at 160×120 pixels (19 fps) with a color depth of 12 bits YCrCb. It has many powerful embedded image processing functions for tracking multi-colored blobs.

So far we have tested this camera successfully on CM-510 and CM-530 using RoboPlus TASK as well as Embedded C on them. This camera performed better on the newer controllers such as OpenCM-9.00 and OpenCM-9.04-B due to the faster MCU clock rate. These application programming projects are described later in Chap. 9.

3.4.5 *GPIO (5-Pin) DMS Sensor*

This sensor uses a triangulation technique and a Position Sensitive Detector to determine the angular displacement of the reflected light beam and thus can compute the distance to the object (http://www.sharpsma.com/webfm_send/1489). It can achieve a longer detection range (10–80 cm) than the AX-S1 and is mostly independent of objects' brightnesses (see Fig. 3.30).

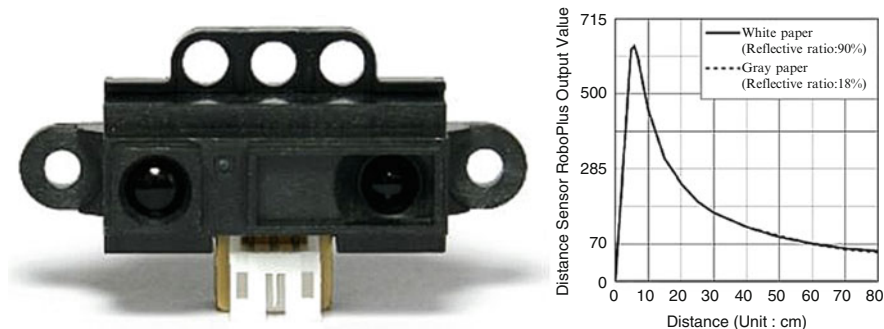


Fig. 3.30 DMS and output vs. distance graph

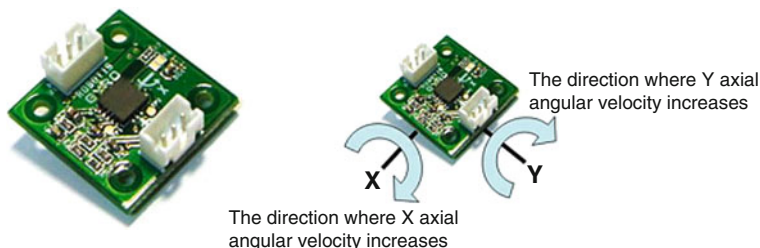


Fig. 3.31 MEMS based Gyro Sensor (GS-12)

3.4.6 GPIO (5-Pin) Gyroscope Sensor GS-12

This sensor (see Fig. 3.31) is a MEMS based 2-Axis gyroscope with digital output values between 45 and 455, corresponding respectively to $-300^\circ/\text{s}$ and $+300^\circ/\text{s}$ (i.e., we are dealing with very fast motion rates for your robot).

It can be used within a Callback routine to help stabilize Humanoid robots doing fast walking moves. It is usually mounted near the Center of Gravity of the robot, and because it is MEMS based, it is immune to electromagnetic interferences from the actuators. Some gyro applications programming are presented in Chap. 9.

3.4.7 Other GPIO (5-Pin) Sensors and Output Devices

When the OLLO system was coming out in 2008, a few GPIO sensors, actuators and output devices also became available: IR, Touch, Servo Motor and LED display (see Fig. 3.32).

The latest crop of GPIO sensors came with the ROBOTIS-MINI in 2014: color, magnetic, temperature, ultrasonic, object detection (see Fig. 3.33).

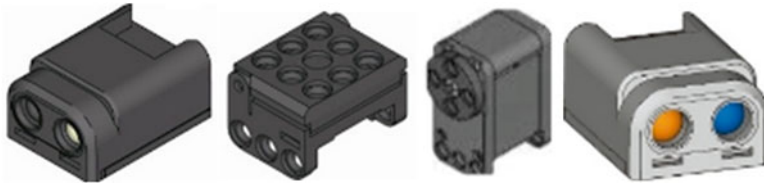


Fig. 3.32 GPIO modules: IR, Touch, Servo Motor & LED display



Fig. 3.33 Color, magnetic, temperature, ultrasonic, object detection sensors

3.4.8 Recent Adaptations of Smart Phone Features

Recently, ROBOTIS started to integrate technologies from Smart mobile devices into “Firmware 2.0” controllers such as the CM-150, CM-200 and OpenCM-9.04 system to obtain a whole host of new features as shown in the screen-captures below (Fig. 3.34):

Thus, in the next few years, we can expect a coming of age of “Personal/Mobile Robotics” applications. But that will have to be the subject of another book!

3.5 Review Questions for Chap. 3

1. Which Firmware group does the CM-530 belong to?
2. Which Firmware group does the CM-200 belong to?
3. Which controller(s) use a mini-jack connector (BSC-10) to connect with the PC?
4. Which CM controller(s) allow simultaneous wired and wireless communications from the PC?
5. How many “wired” interface modules are supported by ROBOTIS? And what are their names?
6. How many “wireless” interface modules are supported by ROBOTIS? And what are their names?
7. Which wireless protocol would open two COM ports on the PC side for each wireless module used?
8. Which module allows direct interfacing to the Dynamixel?
9. Which member of the CM-5XX family of controllers has 128 KB of RAM?
10. Which member of the CM-5XX family of controllers clocks at 72 MHz?

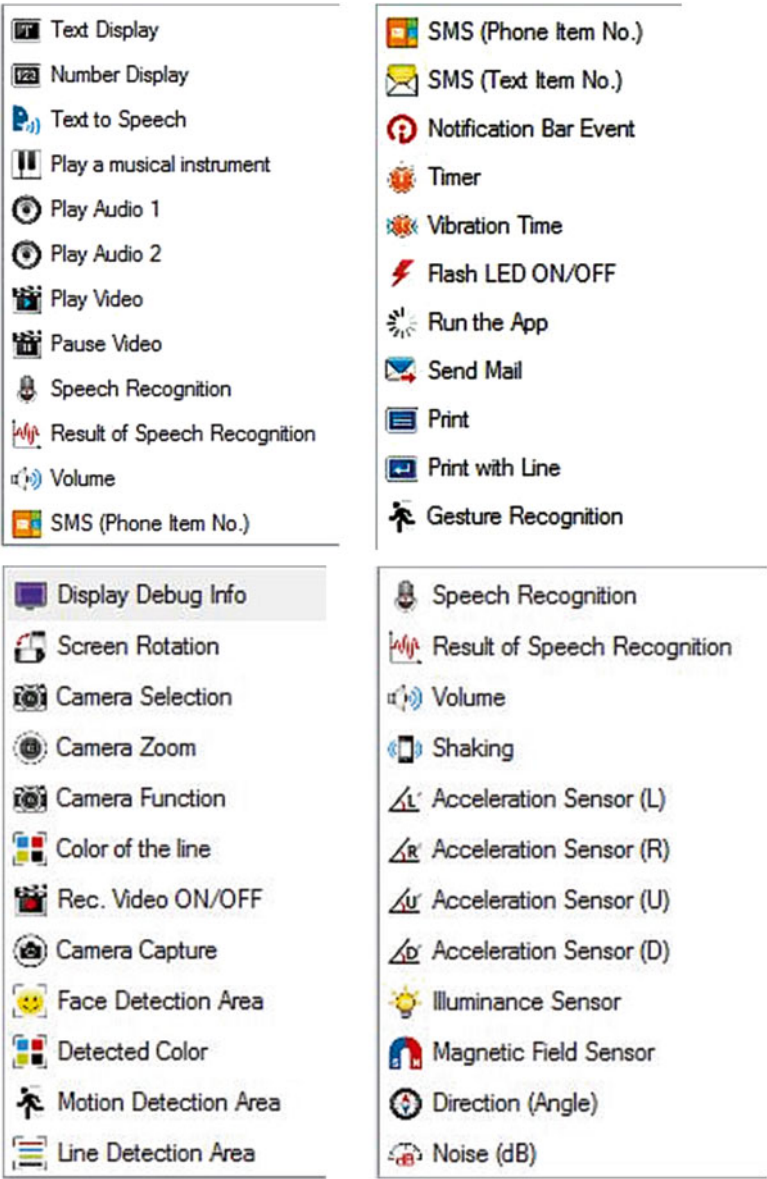


Fig. 3.34 Integration of Smart mobile technologies into R+Task tool (partial list)

- 11. Which controller(s) use the Atmel AVR controller?
- 12. Which controller(s) use an ARM Cortex M3 architecture?
- 13. Which Dynamixel hardware interface uses a 4-pin connector? And which one is using 3-pin connector?
- 14. How many pins does a typical GPIO port have?

15. How many pins does the communication port have? This port is used to connect modules such as LN-101, ZIG-110A or BT-210.
16. On which controller(s) can the user access wired and wireless connections at the same time?
17. Hardware wise and software wise, how can the user connect wirelessly from a PC to a CM-5 based robot?
18. Hardware wise and software wise, how can the user connect wirelessly from a PC to a CM-510 based robot?
19. Hardware wise and software wise, how can the user connect wirelessly from a PC to a CM-530 based robot?
20. Hardware wise and software wise, how can the user connect wirelessly from a PC to an OpenCM-9.04 based robot?
21. How many ways can the user access the Virtual RC-100 Controller? From which tools?
22. Which sensor modules can be used to measure distances?
23. Which controller(s) have a built-in microphone? And which one(s) do not?
24. Which version(s) of the OpenCM-9.04 controller can be used with the RoboPlus Software Suite?
25. Which controller(s) have real-time debugging capabilities (JTAG/SWD)?
26. What are the difference(s) between the three versions A-B-C of the OpenCM-9.04?
27. How many serial communication ports can the OpenCM9.04 support?
28. Which sensor(s) interface to the GPIO ports of the ROBOTIS controllers?
29. Which sensor(s) are used to acquire positional data of a typical actuator when it is operating, such as the AX-12, MX-28 or XL-320?
30. List three sensors that are Dynamixel-compliant.
31. List three actuators that are Dynamixel-compliant.
32. List three sensors that are designed for use on the GPIO ports.
33. Which sensor module has two programmable ranges for its distance measurement task?
34. From a hardware circuitry point of view, how can a servo motor switch its rotation direction?
35. How would a user design a system that can control DYNAMIXEL-PRO actuators?
36. Which Dynamixel actuators can be controlled with a PID approach?
37. What does the specification “W” stand for in the naming of ROBOTIS actuators?
38. What is the hardware refresh cycle time for Dynamixels?
39. What is the type of the physical device used to transduce the angular position of the AX-12 actuator?
40. What is the type of the physical device used to transduce the angular position of the MX-28 actuator?
41. Why does the AX-S1 have a reverse-response to the distance when objects are too close to its NIR-LED sensing element?

42. What is the minimum time interval between sound claps that the AX-S1 needs in order to distinguish the sound claps as distinct audio events?
43. What are the current options for a user who wants to measure inertia-related parameters such as accelerations and rotational rates?
44. What are the current options for a user who wants to measure contact pressures/forces between the robot and the supporting surface?
45. What embedded vision hardware is current available for the CM-5XX and OpenCM systems?

References

Clark D, Owings D (2003) Building robot drive trains. McGraw-Hill, New York
Kanniah J et al (2014) Practical robot design: game playing robots. CRC, Boca Raton

Chapter 4

Software Tools

In this chapter, the goal is to go over the main software tools provided by ROBOTIS using a simple demonstration system having two servo motors but swapping out different controllers (CM-5, CM-510, CM-530 and CM-9.04-B/C) as needed.

ROBOTIS provides their software tools for free and most tools have proprietary source codes and/or firmware, except for the ROBOTIS OpenCM IDE for the CM-9.04-A/B/C systems (which is based on Arduino). Officially, ROBOTIS has not released a “roadmap” for their software tools, so things are still confusing at times even for a long-time practitioner like the author (especially when new products get released), so I’ll do my best to help the reader in sorting what to use to do what and under which circumstances.

Historically speaking, the very first ROBOTIS software suite was the BIOLOID suite (c. 2005) with “Behavior Control Programmer” and “Motion Editor”. It got replaced with RoboPlus V.1.0.8 in 2009 along with the introduction of BIOLOID PREMIUM.

Currently at version 1.1.3.0 (7/8/2014), the RoboPlus suite runs on MS Windows only or “mostly” on a good emulator of it such as Parallels, Fusion, WINE, Boot Camp, etc.... It is available for download at (http://www.robotis.com/xen/download_en/1132559). When installed, it is a portal to five software tools (Task, Manager, Motion, Terminal and Dynamixel Wizard) as shown in Fig. 4.1. Furthermore, once installed, each of these five tools will get automatic updates on their own whenever ROBOTIS pushes them out.

Next, there are currently two “new-generation” software packages for MS Windows (or emulators): R+ Design (V.1.1.5) and R+ Motion (V.2.2.4)—not to be confused with the previous RoboPlus Motion tool (which can be considered as V.1). They are available for download at http://en.robotis.com/BlueAD/board.php?bbs_id=downloads&page=1&key=&keyword=&bbs_opt1=&scate. After installation, they are also set to be automatically updated when available (see Fig. 4.2).

Electronic supplementary material: The online version of this chapter (doi:[10.1007/978-3-319-20418-5_4](https://doi.org/10.1007/978-3-319-20418-5_4)) contains supplementary material, which is available to authorized users.

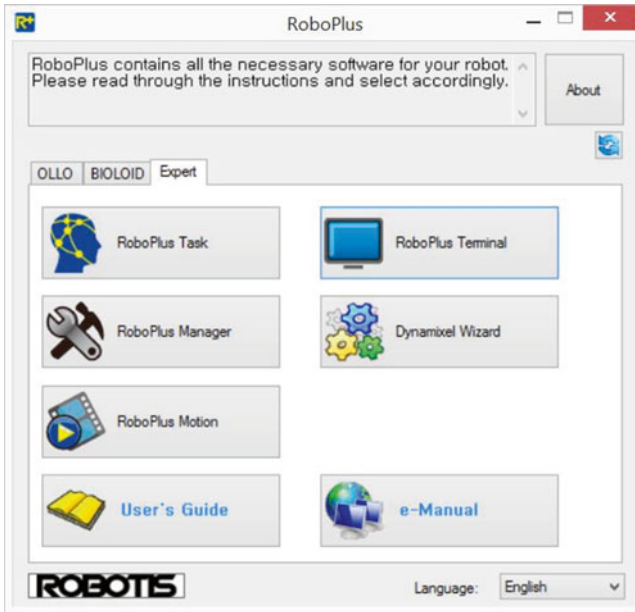


Fig. 4.1 RoboPlus portal

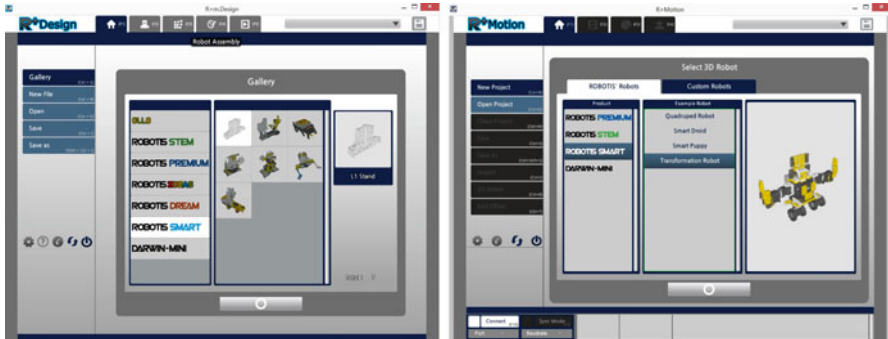


Fig. 4.2 R+ Design and R+ Motion

There are also mobile versions of these software tools (Android OS only for now) which are available for download at Google Play (<https://play.google.com/store/search?q=robotis&c=apps&hl=en>, see Fig. 4.3). These applications, especially R+m.Task and R+m.Motion, leverage extensively on existing SMART technologies available in current mobile devices such as phones and tablets. Although these mobile tools are quite interesting implementations, they are outside the scope of this book.

In the Embedded C area, there are two options for the users. Option 1 is more for professional programmers as it requires quite a bit of hardware and software

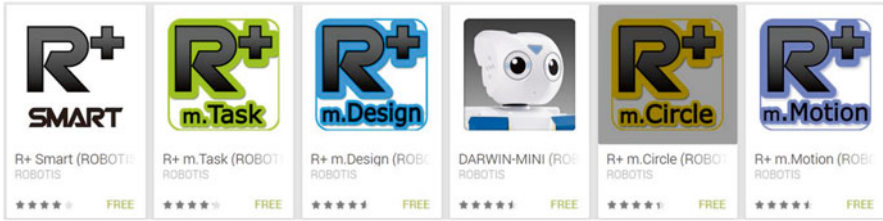


Fig. 4.3 Android's apps of ROBOTIS software tools

knowledge from the user and it uses more complex tool chains based on compilers provided by the manufacturers of the microcontrollers themselves. These software tools, installation procedures and some tutorials are provided at the following link (http://support.robotis.com/en/software/embedded_c_main.htm) and they are designed for the CM-510/CM-700 and CM-530 only. Option 2 has a lower technical bar for entry as it is based on the Arduino interface and it is called OpenCM IDE which is available for download at (http://www.robotis.com/x/download_en/633740). There is also a learning community for the OpenCM system at (http://www.robotsource.org/bd.php?bt=forum_CM9DeveloperWorld). Technical manuals and tutorial materials for OpenCM were provided previously in Chap. 2 as ZIP files. Both Embedded C options will be discussed further in Chap. 10.

4.1 Dynamixel Wizard Tool

You should not have to use this tool at all if you recently bought your new controllers and Dynamixel actuators or sensors, as they would have the latest firmware installed from the factory. In general, the Dynamixel's firmware is updated via the Dynamixel Wizard and the Controller's firmware is updated via the Manager tool, but see Sect. 4.1.2.1 for specific procedures for the OpenCM-9.04-C.

First you will need to acquire the USB2Dynamixel module (http://www.robotis-shop-en.com/?act=shop_en.goods_view&GS=1289&GC=GD080300) or the LN-101 module (http://www.robotis-shop-en.com/?act=shop_en.goods_view&GS=1277&GC=GD080300) depending on the type of controllers used (see chart at http://support.robotis.com/en/techsupport_eng.htm#product/auxdevice/controller_main.htm).

4.1.1 TTL (3-Pin) and RS-485 (4-Pin) Dynamixels

Your best option is to go with the USB2Dynamixel module and you also need to arrange for independent power of the Dynamixels via the SMPS2Dynamixel (http://www.robotis-shop-en.com/?act=shop_en.goods_view&GS=1267&GC=GD080303)

Fig. 4.4 Typical setup for Dynamixel firmware update using a CM-530 and two AX-12As



or just use the controller module as the power source (see Fig. 4.4 for an example setup with a CM-530 and two AX-12As). If you are using a CM-5, CM-510 or CM-700, you can use the same setup and just swap out the controller.

Figure 4.4 shows that the LiPo battery is powering the CM-530 controller and indirectly the AX-12As via the 3-pin Dynamixel bus, while the PC (i.e. Dynamixel Wizard) communicates to the AX-12As via the chain “USB»USB2Dynamixel»AX-12As”. Please note that the side switch on the USB2Dynamixel module needs to be slid into the TTL (or RS-485) position (as appropriate) for this setup to work properly.

If you are only “updating” the firmware of your Dynamixels, you can string them up in a daisy-chain fashion as shown in Fig. 4.4, as the Wizard tool can update several Dynamixels (of the same type, to be on the safe side) during the same cycle. However, if something “real bad” had happened to a particular Dynamixel, and as part of the troubleshooting process you are trying to see if “recovering” the firmware on it will help, then you will need to hook up only ONE Dynamixel at a time. Procedures for updating, recovering and testing firmware are accessible at this link (http://support.robotis.com/en/software/roboplus/dynamixel_wizard.htm). For a more recent video of the process, please review enclosed video file “Video 4.1”. If you are “recovering” successfully, remember to set the Dynamixel ID back to the one you were using before, as the “recovery” process will reset the ID to “1”.

4.1.2 XL-TTL (3-Pin) Dynamixels

Currently, the only XL-TTL Dynamixel in existence is the XL-320 operating with the OpenCM-9.04-A/B/C controllers. At the time of writing of this book, ROBOTIS was still working on their RoboPlus tools to work with the OpenCM systems,

thus the procedures described herein were obtained from experimentation by the author. So please be aware that, at a later time, final and official ROBOTIS procedures may be different.

Due to the difference in firmware used on the A/B version and the C version, the respective firmware update processes were also different.

4.1.2.1 OpenCM-9.04-C

The 9.04-C controller’s firmware and the firmware of XL Dynamixels attached to it can be updated at the same time via the 4-pin communication port. The user has several options for communications hardware, such as “wired” via the LN-101 or “wireless” via ZIG-110 and BT-110/210. The LN-101 route is the most reliable for this rather delicate operation, but the reader should try other options just for learning experiences.

Figure 4.5 describes a setup using an LN-101 with a 9.04-C and 2 XL-320 servos.

Once the “Dynamixel Wizard” tool is started (currently at V.1.0.19.5), the user can click on the icon for “XL-320 Firmware Management” (see Fig. 4.6) and choose either “Update” or “Recovery” depending on the need and then follow the on-screen instructions (see video clip “Video 4.2” for more details).

Please note that this tool will additionally update the firmware of the 9.04-C controller to its most current version.

Fig. 4.5 Typical setup for Dynamixel firmware update using an OpenCM-9.04-C

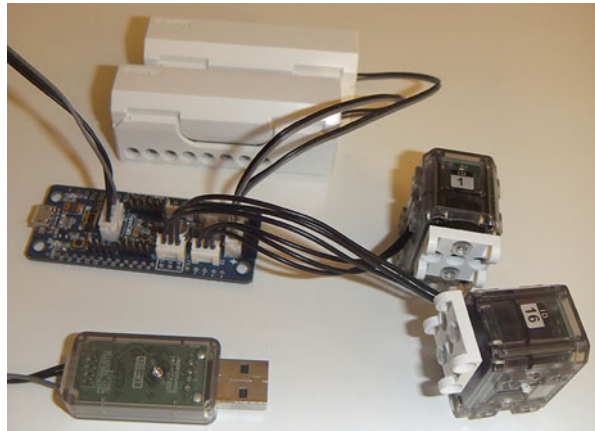


Fig. 4.6 Pull-down menu for the “XL-320 Firmware Management” icon

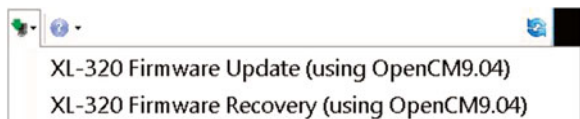
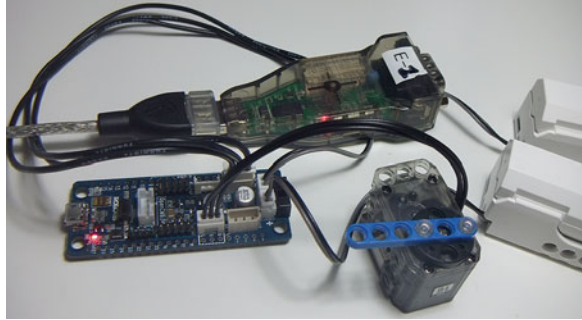


Fig. 4.7 Typical setup for Dynamixel firmware update using an OpenCM-9.04-B



4.1.2.2 OpenCM-9.04-B

At present, if the user only has the OpenCM-9.04-B controller, the Dynamixel firmware update process is much more complex because the B version does not have a “nice” firmware to interact with the previous “XL-320 Firmware Management” tool (Sect. 4.1.2.1). In a way, the user will have to use similar steps shown in Sect. 4.1.1 with the USB2Dynamixel module. Figure 4.7 shows the required hardware setup:

- The USB2Dynamixel module serves as the interface between the PC’s USB port and the Dynamixel Port on the OpenCM-9.04-B (USB2Dynamixel’s side switch set on TTL as per Sect. 4.1.1).
- The OpenCM-9.04-B controller acts a common Dynamixel bus because it has both AX/MX-TTL and XL-TTL connectors. It also serves as the power source for the XL-320 actuators.

This setup **also requires** that the “DxlTosser” sketch be preloaded and running on the 9.04-B (see Chap. 10 for more details). Then ones can get the Dynamixel Wizard tool started and apply similar procedures described in Sect. 4.1.1 for the AX/MX-TTL Dynamixels to the XL-320 actuators, i.e. “updating” several Dynamixels of the same type or “recovering” only 1 Dynamixel (see “Video 4.3” for more details).

Updating the 9.04-B controller’s firmware would involve JTAG/SWD procedures that are not yet released by ROBOTIS.

4.2 Manager Tool

4.2.1 CM-5, CM-510, CM-530

The Manager tool, currently at V.1.0.33.2, is fully functional for the CM-5, CM-510 and CM-530 controllers and the user **must choose** wired connections via the USB2Dynamixel \gg BSC-10 (CM-5 and CM-510) or USB-mini cable (CM-530).

Fig. 4.8 “Bal’Act”
Humanoid A robot



The video file “Video 4.4” highlights how Manager can perform a quick check on a Bioloid Premium Humanoid A equipped with some extra sensors such as AX-S20, Gyro and Foot Pressure Sensors (see Fig. 4.8).

More on-line resources for the Manager tool can be found at http://support.robotis.com/en/software/roboplus/roboplus_manager_main.htm.

4.2.2 *OpenCM-9.04-A/B/C*

First, the OpenCM-9.04-A/B controllers were designed to work with the ROBOTIS Arduino-like IDE so they do not work with the Manager tool at all.

At the time of writing for this book, the Manager tool (V.1.0.33.2) was not yet fully functional with the OpenCM-9.04-C system, thus it could **only update** the 9.04-C’s firmware (which could be done via the Dynamixel Wizard tool also—see Sect. 4.1.2.1). Through experimentations, the author has determined that the LN-101 module should be used for the most reliable results with this procedure.

Figure 4.9 shows a hardware setup needed to update the firmware of an OpenCM-9.04-C controller using an LN-101.

The next step is to start the Manager tool. At the main menu bar, choose the correct COM port corresponding to the LN-101 and click on the “Controller Firmware Management” icon (left of the “?” icon) and follow on-screen instructions to get the 9.04-C controller updated to its latest firmware version (see video file “Video 4.5” for more details).

If the user had bought a ROBOTIS(DARWIN-MINI) kit which came with a BT-210, the above procedure also worked well, as long as the OUT-GOING COM port was chosen for the connection between the PC and the BT-210.

Fig. 4.9 Typical setup for controller firmware update for an OpenCM-9.04-C



If the user had bought a “loose” OpenCM-9.04-C (<http://www.robotis.us/opencm9-04-c-with-onboard-xl-type-connectors/>) which came with a USB-to-micro cable, the above procedure also worked quite satisfactorily when using the appropriate COM port.

4.3 Task Tool

The TASK tool, currently at V.1.1.2.4, is a cross-over between a standard text-based IDE such as Eclipse or ROBOTIS IDE and a pure icon-based such as Lego NXT to help beginners with a more controlled syntax. It has all the standard control structures (sequential, repetition and conditional) and provides variables and functions definition. It even supports 1 Callback function, but it does not support variable arrays.

Since 2013, the TASK tool is well documented in the Software Programming Guide included in the Bioloid Premium kit. If you got an earlier edition of this kit that did not have this user’s guide, I would recommend first-time users to get it at http://www.robotis-shop-en.com/?act=shop_en_goods_view&GS=1486&GC=GD080400. The ROBOTIS e-Manual web site also has much technical information for the Task tool at http://support.robotis.com/en/software/roboplus/roboplus_task_main.htm. Thus instead of repeating all this information here, I would like to share a “comparative study” across different controllers using a demonstration setup with two AX-12As or XL-320s using “wired” and “wireless” (ZigBee and BlueTooth) communications using the Virtual RC-100.

4.3.1 CM-5, CM-510, CM-530

The demonstration setups for the CM-5XX family is shown in Fig. 4.10.

The test TASK code (TestVRC-100_CMs.tsk) is shown in Fig. 4.11.



Fig. 4.10 Demonstration setups for TASK “comparative study” (left to right—CM-5, CM-510 and CM-530)

```

1  START PROGRAM
2  {
3      ENDLESS LOOP
4      {
5          WAIT WHILE ( Remocon Data Received == FALSE )
6          Button = Remocon RXD
7
8          IF ( Button == U )
9          {
10             ID[1]: Goal Position = 1023
11             ID[16]: Goal Position = 1023
12         }
13         ELSE IF ( Button == D )
14         {
15             ID[1]: Goal Position = 0
16             ID[16]: Goal Position = 0
17         }
18         ELSE IF ( Button == -- )
19         {
20             ID[1]: Goal Position = 512
21             ID[16]: Goal Position = 512
22         }
23     }
24 }

```

Fig. 4.11 Demonstration test code (TestVRC-100_CMs.tsk)

It is quite simple:

- Just an overall `Endless_Loop` with an initial `Wait_While` loop checking on whether the Virtual RC-100's buttons had been pushed or not (line 5).
- When a button had been pushed, check it to see if it was:
 - “Up” then set the `Goal_Position` for servos 1 and 16 to “1023” (lines 8–12).
 - “Down” then set the `Goal_Position` for servos 1 and 16 to “0” (lines 13–17).
 - “Nothing”, i.e. upon release of any button of the RC-100's buttons, tell servos 1 and 16 to go to `Goal_Position` “512” (lines 18–22). The author chose to treat this condition explicitly because it was known that the RC-100 would send the “0” signal (statement 18) only **once**, thus any “worthy” communication hardware/software implementation would have to be able to pick up this one-time event. This feature was used to evaluate the performance between wired and wireless communications as shown in sections below.

The video file “Video 4.6” showed how this test code was performing on a CM-5 using a “wired” connection (i.e. `USB` \gg `USB2Dynamixel` \gg `BSC-10` \gg `CM-5`) or a “wireless” connection (i.e. `USB` \gg `USB2Dynamixel` \gg `Zig2Serial` \gg `ZIG-100-1` \gg `ZIG-100-2` on `CM-5` \gg `CM-5`). Both options worked well without any discernable performance issue.

This test code was also tried on a similar system with a CM-510 using “wired” and “wireless” connections. For a CM-510, there were three “wireless” options: `ZIG-110`, `BT-110` or `BT-210`. The video file “Video 4.7” showed that all three wireless options were functional but there were some subtle performance differences:

- ZigBee connections were made within 1 s of executing of the program, while BT connections could take up to 3–4 s to be done, thus users need to take these BT delays into account for their own applications.
- ZigBee data buffers seemed to be updated much more efficiently than BT data buffers, as BT systems seemed to be “stuck” when a particular VRC-100 button was held in for 2–3 s. It is not known whether this is an issue on the PC side or on the BT modules side, or that this is an expected behavior of Bluetooth protocols.
- BT-210's performance was less than BT-110's in the above test conditions.

When the same tests were performed on a CM-530 system (“wired” plus the previous three “wireless” options), all wired and wireless connections were functional but the same issues discussed previously were also found on the CM-530 (see video file “Video 4.8”).

4.3.2 CM-9.04-C

To complete the comparative study, the same TASK code was tested on an OpenCM-9.04-C system with the same four communication protocols as in Sect. 4.3.1, but with XL-320 servos instead:

- Wired, through LN-101.
- Wireless, through ZIG-110, BT-110 and BT-210.

The video file “Video 4.9” showed that the previous wireless issues existed for the OpenCM-9.04-C also.

(Note: since the V1.1.2.4 update for TASK (9/5/2014), the previous BlueTooth issues are resolved satisfactorily).

In conclusion, at present, ZigBee is the best performing wireless option for controlling ROBOTIS robotic systems, however it is also recognized that BlueTooth is more widely implemented on PCs and mobile devices thus requiring no extra hardware for the users to buy. It was also shown that a TSK code is portable across existing RoboPlus-compatible platforms as long as the “correct” controller was chosen before compiling and downloading the codes.

4.4 Motion Tools (V.1 and V.2)

The TASK tool is well suited for reading and writing data/commands from/to a small number of actuators and sensors as on a wheeled robot, as the inherently “sequential” execution of those commands (line by line as shown in Fig. 4.11) still has an acceptable performance. However on a humanoid robot with 18 actuators that have to work together and “simultaneously” in order to perform a maneuver such as walking, a different way of generating and executing sets of actuator goal positions in “parallel” on those 18 actuators in a timely manner has to be devised.

For this purpose, ROBOTIS created the Motion tools, from the original BIOLOID Motion Editor (c. 2005) to the RoboPlus Motion V.1 (c. 2009), to the latest R+ Motion V.2 (c. 2014) (see Fig. 4.12). Conceptually, these Motion tools were based on cartoons (or frames) animation which is essentially the re-playing of a sequence of poses of the considered character (robot) within a time line, with slight variations in each pose to create the illusion (perception) of motion.

The BIOLOID Motion Editor only provided front views of a 3-D robot, and they were shown “relative” to the main controller and not to the ground surface. With RoboPlus Motion V.1, 3-D graphics of the robot are available (still relative to the controller/body) but were used more for animation check of the robot poses, rather than for creating and editing them. Since 2013, the Software Programming Guide (SPG) included in the Bioloid Premium kit has very good tutorial materials

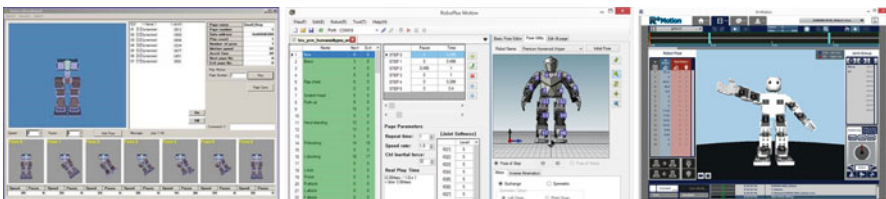


Fig. 4.12 Evolution of the Motion tools from 2005 to 2014



Fig. 4.13 Key features of R+ Motion V.2

for learning how to use the Motion tool (V.1). I would recommend beginners to obtain it from http://www.robotis-shop-en.com/?act=shop_en.goods_view&GS=1486&GC=GD080400 and use it well. The ROBOTIS e-Manual also has much information for the Motion V.1 tool (http://support.robotis.com/en/software/robo-plus/roboplus_motion_main.htm), but the materials in the SPG is better suited for self-learners. The enclosed video file “Video 4.10” shows how to use RoboPlus Motion V.1 on a PREMIUM GERWALK robot. Additional applications of this tool will be provided in Chap. 6.

R+ Motion V.2 was released in Spring 2014 and it supports PREMIUM, STEM, SMART & ROBOTIS-MINI systems (currently it is at V.2.2.4). RoboPlus V.1 is still available to support robots using CM-5 and CM-510, but most likely it will be phased out in a few years along with the CM-5 and CM-510 controllers. Three key features of R+ Motion V.2 should be noted (see Fig. 4.13):

1. The display of a global time line for each “motion-unit”.
2. The use of a “physics” engine so that the robot 3-D frame can be manipulated relative to the “ground” surface, and not with respect to the robot’s body as in V.1. However this “physics” engine won’t be able to accommodate highly acrobatic moves like body flipping or rolling!
3. The robot motions can be edited and executed on an existing 3-D model without the need of the real robot actually being connected to the PC or mobile device during that time.
4. On fast graphics display hardware, the graphics engine will be able to synchronize the simulated robot moves to the real robot moves in the physical world.

The web-based user manual for the R+ Motion tool V.2 is available at http://support.robotis.com/en/software/roboplus2/r+motion2/rplus_motion2.htm. Enclosed is a video illustrating some basic uses of R+ Motion V.2 on an OpenCM-9.04-C with 2 XL-320 servos (Video 4.11). More applications of this tool will be shown in Chap. 11.

4.5 R+ Design Tool

Spring 2014 also saw the first release of the R+ Design tool (V.1.1) for MS Windows PCs (see Fig. 4.14), but it was originally designed for mobile systems (currently Android OS only) and for the OLLO system as part of the suite of R+m.Task and R+m.Design (c. 2012).

The current R+ Design V.1.1.5 supports the following systems: OLLO, STEM, PREMIUM, PLAY, IDEAS, DREAM, SMART and ROBOTIS-MINI. At present, R+ Design only supports 3-D design and assembly of ROBOTIS parts, but the R+ Design and Motion tools together show ROBOTIS' vision of integration from design to simulation/control/testing (on the computer) for individual users as well as for communities of robotics designers and instructors. At present, those activities are beginning via websites such as RobotSource and STEAM Education Association (http://www.robotsource.org/bs/bd.php?bt=proj_ucrgallery_1 and http://www.steamcup.org/new/?mid=main_eng#).

Currently, March 2015, only the Korean version of the user manual for R+ Design is available at http://support.robotis.com/ko/software/r+design_main.htm.

Personally, I have used R+ Design on OLLO kits with my younger students (5–6 years old) and they definitely preferred R+ Design over the instruction pages of the paper-based manuals, because they can “see” the 3-D assemblies from different view angles on the computer screen. However, there was one issue that required the students to refer back to the paper-based manuals and it was about cable routing (at present, only the paper manuals have illustrations of “proper” cable routing during assembly).



Fig. 4.14 R+ Design tool with a SMART robot

4.6 “If I Were to Restart ...”

Recapping Chaps. 3 and 4 and if I were to restart my robotics journey at the present time, I would start with the BIOLOID PREMIUM system using the CM-530 controller. Next I would replace the CM-530 with the OpenCM-9.04-C controller combined with the OpenCM-485 shield so that I could keep using my 12 V LiPo battery and the AX/MX Dynamixels. But now I could access multi-media features provided by Android tablets and smart phones via R+ V.2 software suite. Later, I could switch to OpenCM IDE for closer access to the ARM controller.

4.7 Review Questions for Chap. 4

1. Which option(s) does a user have to run ROBOTIS software on Mac OS based computers?
2. What are the software tools that are available with the ROBOPLUS software suite on Windows platforms? Or on Android mobile platforms?
3. What tasks can the MANAGER tool provide?
4. What tasks can the TASK tool provide?
5. What tasks can the DYNAMIXEL WIZARD tool perform for the user in conjunction with a USB2Dynamixel or LN-101 module?
6. Which controllers can have their firmware updated using the MANAGER tool?
7. How can firmware update/recovery be performed for a Dynamixel actuator?
8. Which way(s) can be used to set the IDs of the ZigBee modules? Via MANAGER tool? Or via TASK tool?
9. From which tool can the user perform ZigBee management tasks with the ZIG2SERIAL module?
10. Which kind of signal (if any) does the “physical” RC-100 Remote Controller send out when the user releases one of its buttons?
11. Which kind of signal (if any) does the “virtual” RC-100 Remote Controller send out when the user releases one of its buttons?
12. What are the differences between versions 1 and 2 for ROBOTIS’ MOTION tools?
13. What graphics engine is the R+MOTION tool V.2 based on?
14. List key features of the R+MOTION V.2 tool.
15. What can the R+DESIGN tool be used for?
16. Which ROBOTIS robotics system does the R+ DESIGN tool support?
17. What are the Embedded C options for the user?

Chapter 5

Foundational Concepts

For this chapter, the assumptions are that the reader is a beginner to robotics but has some exposure to computer programming. This chapter's main topics are listed below:

- “Sense-Think-Act” paradigm and illustration of its diverse interpretations in the design of hardware and software systems using selected robotics systems.
- “Sequence Commander” CarBot to explain “Sequential-Repetition-Selection” controls and “Functional Decomposition” concepts used in basic robotic programming.
- “Smart Avoider” CarBot to illustrate “reactive” and “behavior-based” control architectures.
- “Line Tracer” CarBot to demonstrate “timed” and “conditional” maneuvers, the concept of “self-localization” and the feedback between robot design and robot performance tuning.
- Introductory Remote Control concepts using Virtual and Physical remote controllers.

The above topics will be developed using the ROBOTIS MANAGER and TASK tools. For more advanced topics in wheeled robots, the reader is referred to Campion and Chung (2008), Dudek and Jenkin (2010) and Cook (2011).

5.1 “Sense-Think-Act” Paradigm

Winfield (2012) defined a robot as:

1. *an artificial device that can **sense** its environment and **purposefully act** on or in that environment;*
2. *an **embodied** artificial intelligence; or*
3. *a machine that can **autonomously** carry out useful work.*

Electronic supplementary material: The online version of this chapter (doi:[10.1007/978-3-319-20418-5_5](https://doi.org/10.1007/978-3-319-20418-5_5)) contains supplementary material, which is available to authorized users.

Nourbakhsh (2013) pointed out that robotics research and innovation had been inspired from human intelligence which depends on two aspects (Fig. 5.1):

1. a meaningful two-way connection with the world; this connection has inputs termed “perception” and outputs back to the world termed “action” (from our own body or via tools that we create).
2. an internal decision making process termed “cognition” that maps our sensory inputs into deliberate actions.

When translated to the realm of world-robot interactions, we got the familiar “Sense-Think-Act” paradigm (Fig. 5.2).

Ones should note the “endless loop” shown in Figs. 5.1 and 5.2 as this is the fundamental characteristic found in all robotics computer programs. This usually is

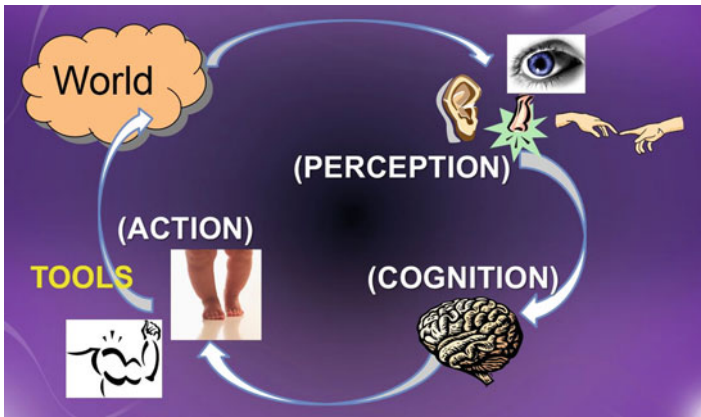


Fig. 5.1 World-human interactions

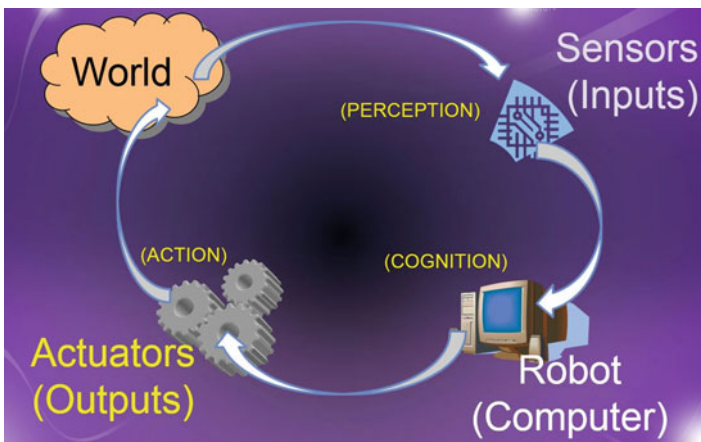


Fig. 5.2 World-robot interactions

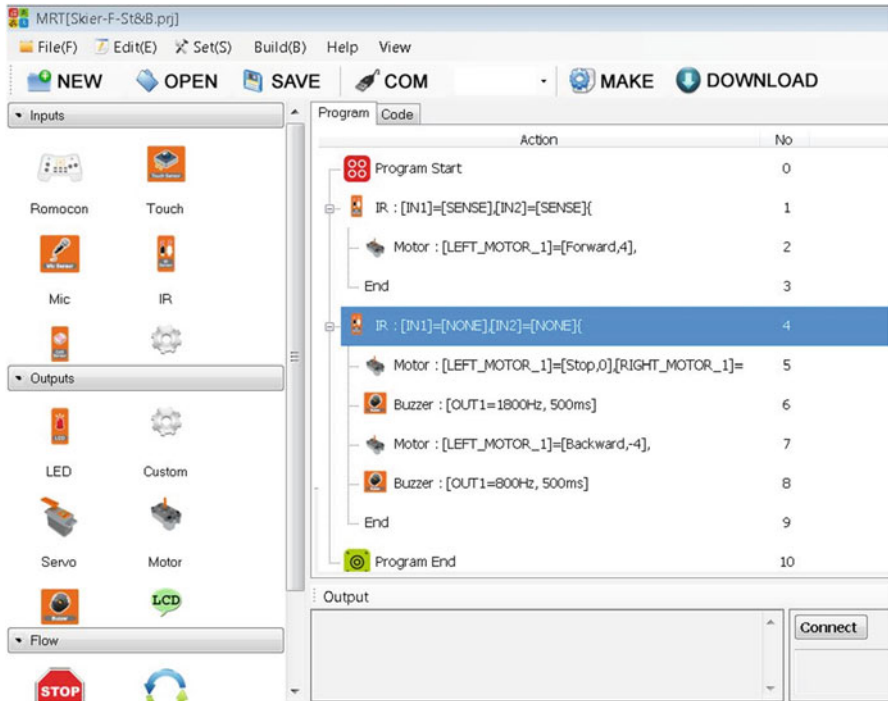


Fig. 5.3 MRT3 system’s software interface

the first “conceptual” obstacle to overcome for beginners, as we all learned how to do basic computer programming by designing and creating programs that would run only once. Some recent robotics systems designed for “very young” roboteers even “hide” this implicit “endless loop”, such as the MRT3 system from My Robot Time (<http://www.myrobottime.com/#!/proudt/c8hd>).

Figure 5.3 was a screen capture of the MRT3 programming interface where a fairly constrained Input»Output approach (i.e. Reactive Control) was used. The interested reader can also watch this video clip describing the software programming aspects of the MRT3 system (Video 5.1).

For years, Lego NXT has been the leader in providing a “drag & drop” graphical programming interface for young robot enthusiasts, but a recent robotics product called ABILIX introduced a new graphical approach using standard flowchart icons (<http://www.abilix.com/en/support.php>) along with a parallel C language development interface (see Fig. 5.4 and video clip Video 5.2).

Another innovative robotics product is called CUBELETS from Modular Robotics (<http://www.modrobotics.com/cubelets/>), and in a way it manages to get rid of the software interface entirely by “transferring” enough of the “Think” capacity into the “Sense” and “Act” components of the standard “Sense-Think-Act” paradigm, so as to enable a second “inner” loop between their “Sense” and “Action” Cubelets (see Fig. 5.5 and video clip Video 5.3).

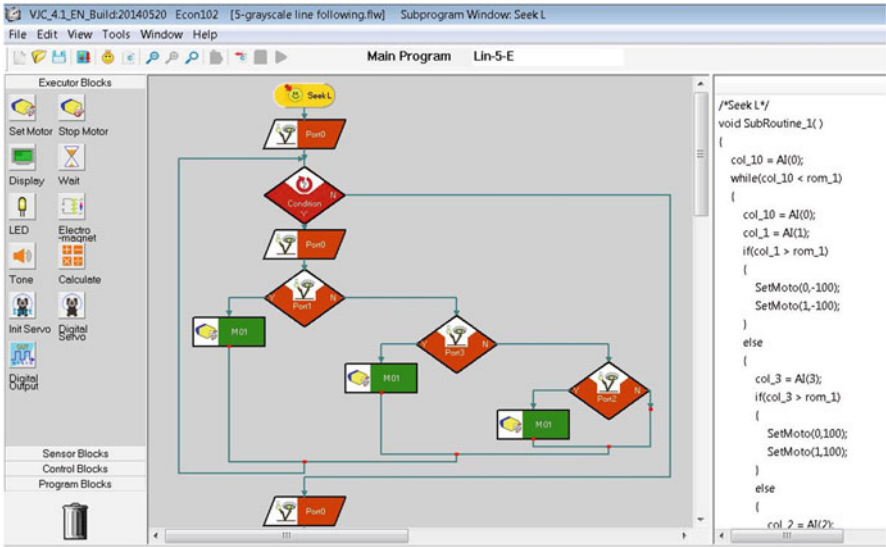


Fig. 5.4 ABILIX system’s software interface

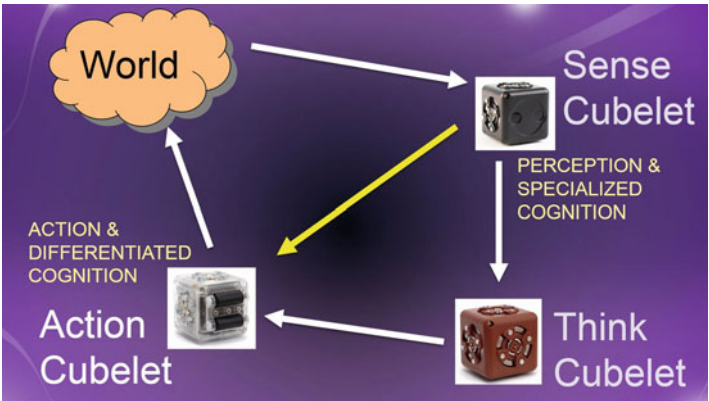


Fig. 5.5 CUBELETS robotics system

The ROBOTIS’ TASK tool could be characterized as a “context-sensitive” text editor and it is closer in programming style to professional tools such as Visual C/ C++ or Eclipse (see Fig. 5.6). Thus it would take comparatively longer to be proficient at it, but then ones can shift to an Arduino-type interface rather quickly (see Chap. 10).

From an end-user point of view, I think that it is good that we have such a range of possible “entry” points into the “robotics” journey to adjust for different maturity and skill levels of every beginner who should realize that this journey can be very far and wide, requiring continual adjustment of the tools that we would use throughout this journey.

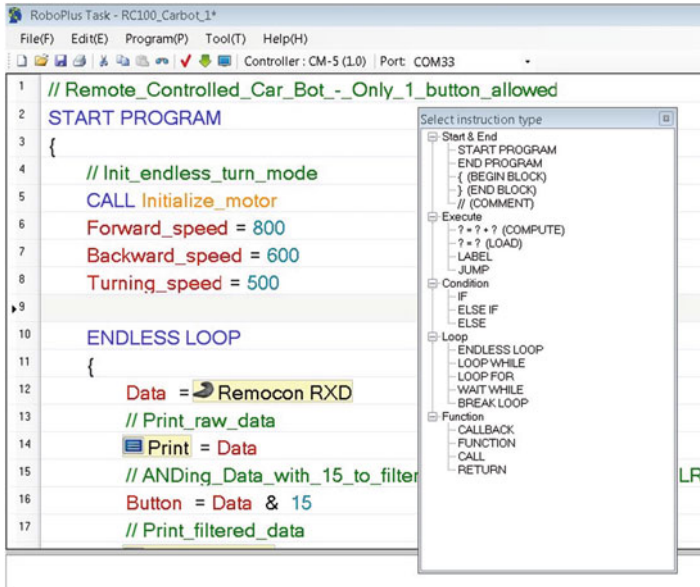


Fig. 5.6 ROBOTIS' TASK tool interface

5.2 Primer for MANAGER and TASK Tools

In practice, MANAGER and TASK work together very well. MANAGER is an efficient tool to do a thorough but quick check on all hardware components with some special operations such as controller firmware update and managing ZigBee communication settings. TASK is the main algorithm development tool supporting all the standard logical structures (sequential, conditional and repetition) and functions, but it does not support arrays and advanced mathematical functions (for those features, the user will need to use Embedded C tools or the OpenCM IDE described in Chap. 10).

5.2.1 MANAGER Capabilities

For this section, a CM-510 CarBot (Fig. 5.7) was chosen to illustrate the use of the MANAGER tool. This CarBot used four AX-12As in wheel mode (ID 1 & 3 on left side; ID 2 & 4 on right side) and two IR sensors, one looking forward (Port 1) and one looking down (Port 2). It also had a ZigBee module (ZIG-110A) set to a 1-to-1 communication mode.

Once MANAGER was executed and connected to the appropriate COM port, the user would see a similar interface to the one shown in Fig. 5.8 (depending on one's

Fig. 5.7 CM-510 CarBot used

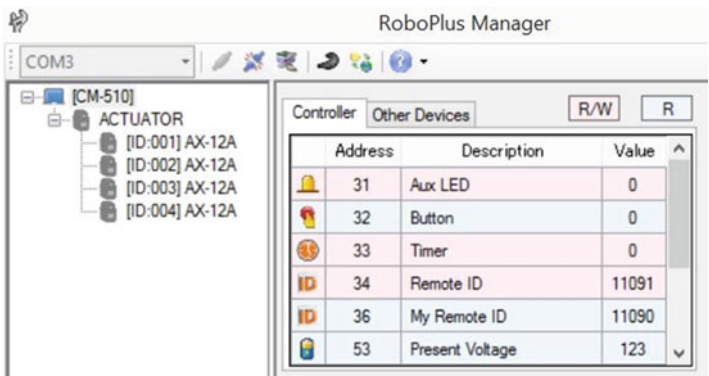
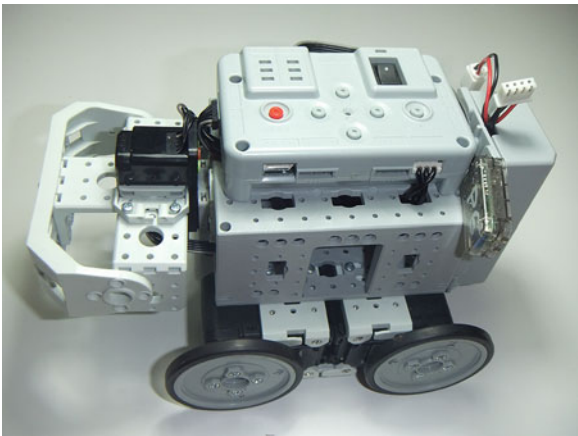


Fig. 5.8 MANAGER Controller Panel (with ZIG-110A in 1-to-1 mode)

actual robot of course). All Dynamixel-compliant (3-pin) actuators and sensors would be listed on the left panel below the Controller item.

In the “Controller” subpanel on the right, the parameter “My Remote ID” (Address 36) corresponded to the actual ZigBee ID (read-only) of the ZIG-110A module used (e.g. 11090). The parameter “Remote ID” (Address 34) was user-editable and it corresponded to the ID of the “other” ZigBee module (e.g. 11091). To make this ZIG-110A module switch to its broadcast mode, the user would need to input “65535” into Address 34. The “other” ZigBee module could be a ZIG-100 module installed inside a CM-5 controller (i.e. another robot) or inside the RC-100 Remote Controller (for more information, please look up the web link http://support.robotis.com/en/product/auxdevice/communication/zigbee_manual.htm). Optionally, the “other” ZigBee module could be another ZIG-110A connected to a CM-510 or CM-530. There are other ZigBee management options which will be discussed in more details in Chap. 8.

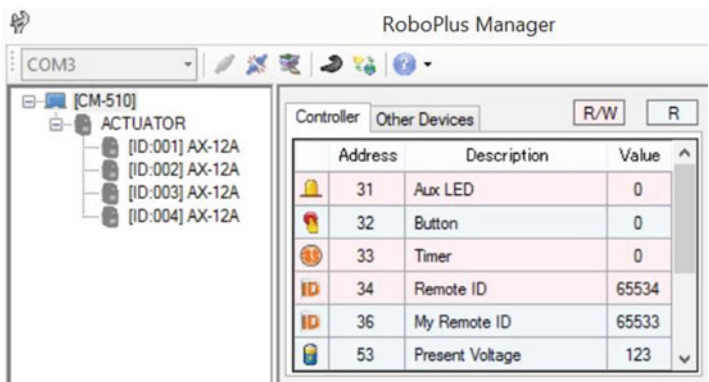


Fig. 5.9 “Controller” subpanel (with BT-110A or BT-210)

Fig. 5.10 “Other Devices” subpanel (with IR sensors on Ports 1 & 2)

Controller		Other Devices		R/W	R
	Address	Control	Value		
Port1	80	IR	12		
Port2	81	IR	321		
Port3	N/A	NONE	0		
▶ Port4	N/A	NONE	0		
Port5	N/A	NONE	0		
Port6	N/A	NONE	0		

If the user happens to be using a BT-110A or BT-210, then the MANAGER Controller Panel would look like Fig. 5.9 and as BlueTooth connections are managed at the Windows Device Manager level, the user should **not** modify Address 34 at all when using BT modules.

In the “Other Devices” subpanel, the user could check on the status of the GPIO (5-pin) actuators and sensors used on the six ports available on the CM-510 and CM-530 (Fig. 5.10). In this particular case, IR sensors were used on Port 1 and Port 2. If MANAGER was used for the FIRST time with any GPIO hardware configuration, the user would have to manually assign the correct sensor to the specific port used for that sensor in order to update the controller flash memory contents.

MANAGER can also be used to test out actuators functions for troubleshooting purposes (Fig. 5.11 and watch the “Video 5.4”). The web link http://support.robotis.com/en/product/dynamixel/ax_series/dxl_ax_actuator.htm has more detailed information on the built-in ROM procedures.

CM-510

ACTUATOR

ID-001 AX-12A

ID-002 AX-12A

ID-003 AX-12A

ID-004 AX-12A

Address	Description	Value
3	ID	1
6	CW Angle Limit (Joint / Wheel ...	0
8	CCW Angle Limit (Joint / Whe...	0
11	the Highest Limit Temperature	70
12	the Lowest Limit Voltage	60
13	the Highest Limit Voltage	140
17	Alarm LED	36
18	Alarm Shutdown	36
24	Torque ON-OFF	0
25	LED	0
26	CW Compliance Margin	1
27	CCW Compliance Margin	1
28	CW Compliance Slope	32
29	CCW Compliance Slope	32
30	Goal Position	883

CM-510

ACTUATOR

ID-001 AX-12A

ID-002 AX-12A

ID-003 AX-12A

ID-004 AX-12A

Address	Description	Value
24	Torque ON-OFF	0
25	LED	0
26	CW Compliance Margin	1
27	CCW Compliance Margin	1
28	CW Compliance Slope	32
29	CCW Compliance Slope	32
30	Goal Position	883
32	Moving Speed	0
34	Goal Torque	1023
36	Present Position	883
38	Present Speed	0
40	Present Load	0
42	Present Voltage	121
43	Present Temperature	38
46	Moving	0

Fig. 5.11 “AX-12” built-in ROM procedures

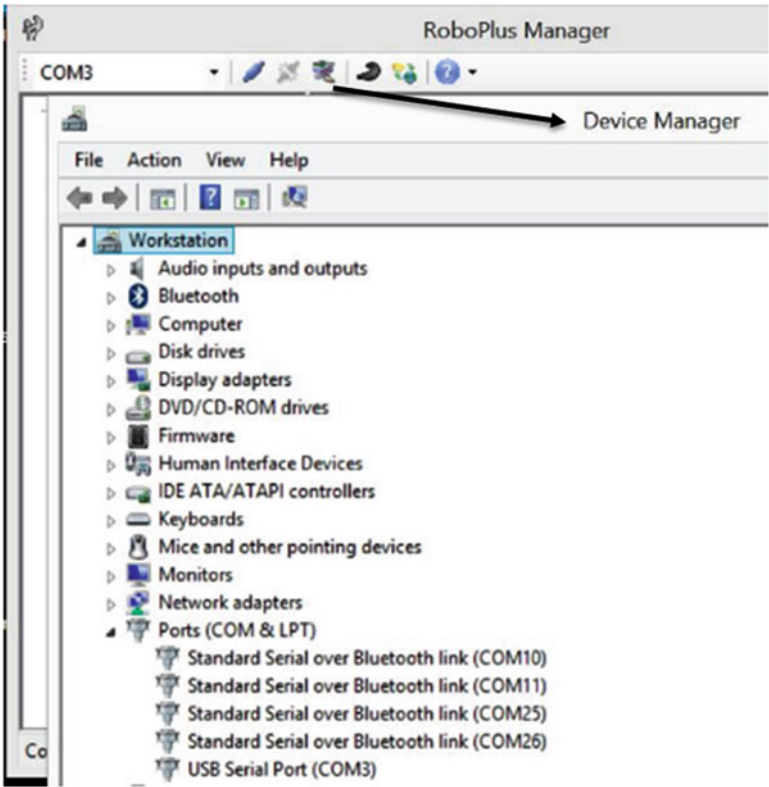


Fig. 5.12 MANAGER’s access to Windows Device Manager

MANAGER also provided access to Windows Device Manager (Fig. 5.12), ZigBee management via the ZIG2SERIAL module (Fig. 5.13 and more details will be forthcoming in Chap. 8), Controller Firmware Management (Fig. 5.14 and previously illustrated in Chap. 4 for OpenCM-9.04 controllers).

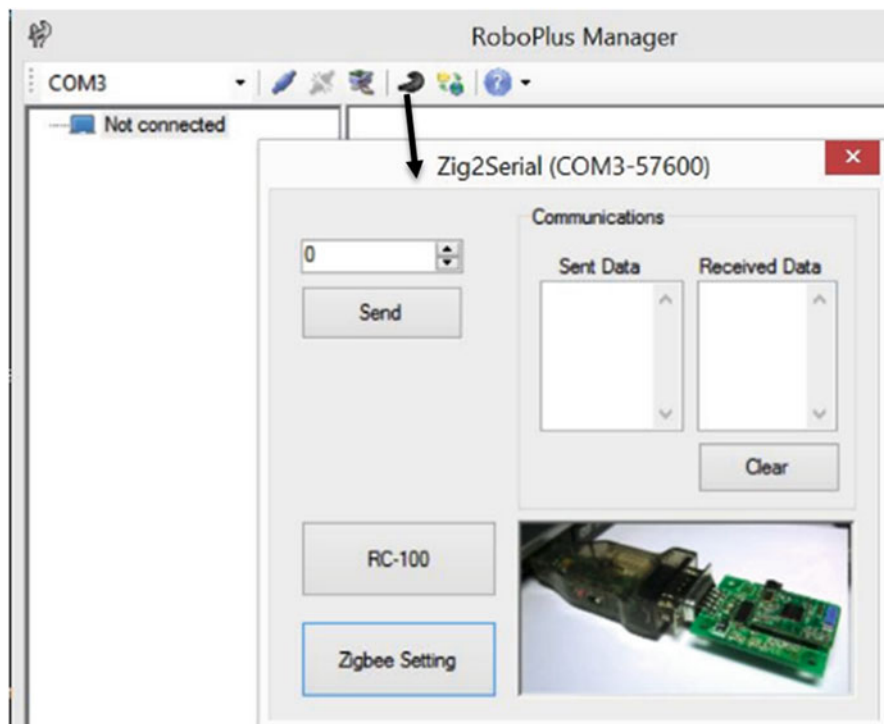


Fig. 5.13 MANAGER's access to ZigBee Management

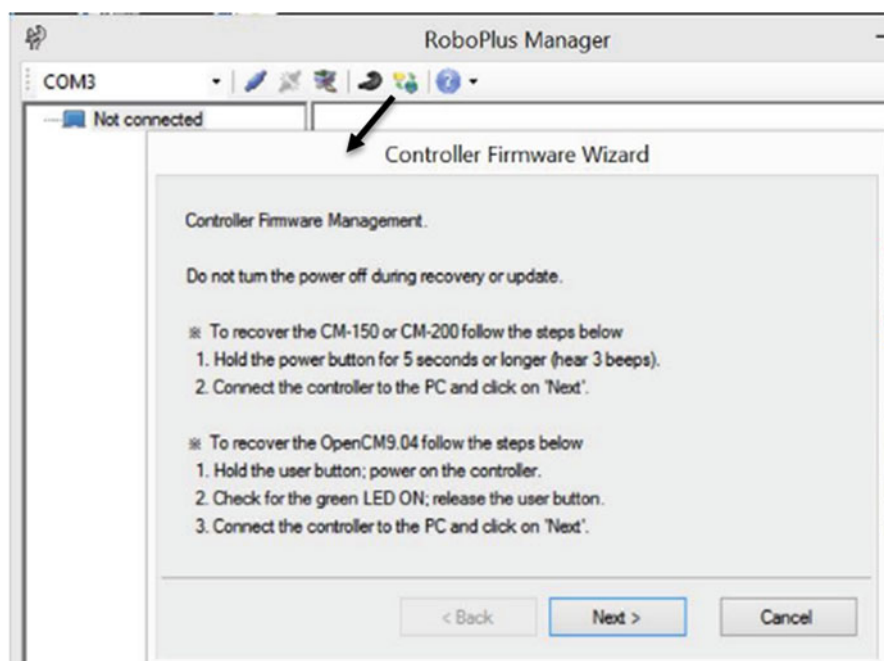


Fig. 5.14 MANAGER's access to Controller Firmware Management

a	b																				
Given Condition >> Appropriate Robot Action	Given Input Sensor(s) >> Activate Appro. Actuator(s)																				
<table border="1"> <thead> <tr> <th>Conditions</th><th>Actions</th></tr> </thead> <tbody> <tr> <td>Condition 1</td><td>Action A</td></tr> <tr> <td>Condition 2</td><td>Action B</td></tr> <tr> <td>Condition 3</td><td>Action C</td></tr> <tr> <td>Condition 4</td><td>Action B</td></tr> </tbody> </table>	Conditions	Actions	Condition 1	Action A	Condition 2	Action B	Condition 3	Action C	Condition 4	Action B	<table border="1"> <thead> <tr> <th>Input Sensors</th><th>Output Actuators</th></tr> </thead> <tbody> <tr> <td>NIR</td><td>DC MOTORS</td></tr> <tr> <td>TOUCH</td><td>SERVO MOTORS</td></tr> <tr> <td>MICROPHONE</td><td>BUZZER-SPEAKER</td></tr> <tr> <td>LIGHT LEVEL</td><td>LEDs</td></tr> </tbody> </table>	Input Sensors	Output Actuators	NIR	DC MOTORS	TOUCH	SERVO MOTORS	MICROPHONE	BUZZER-SPEAKER	LIGHT LEVEL	LEDs
Conditions	Actions																				
Condition 1	Action A																				
Condition 2	Action B																				
Condition 3	Action C																				
Condition 4	Action B																				
Input Sensors	Output Actuators																				
NIR	DC MOTORS																				
TOUCH	SERVO MOTORS																				
MICROPHONE	BUZZER-SPEAKER																				
LIGHT LEVEL	LEDs																				

Fig. 5.15 (a) Condition|Action table; (b) Sensor|Actuator table

5.2.2 Basic TASK Usage

If you happen to be an absolute beginner with the TASK tool, please watch the “Video 5.5” first before reading on the rest of this section. This video demonstrated the basic steps needed from creating a TSK program from scratch to running it on a CM-510 CarBot (Fig. 5.7). It showed how to declare parameters and use printing facilities and also discussed the use of control structures (sequence, loops and conditions). Additionally, it showed how to use functions to modularize coding. Please also refer to the example code files “IR-Sensors-1.tsk” and “IR-Sensors-2.tsk”.

The next example, “IR-Motor.tsk”, illustrated the closing of the “Sense-Think-Act” loop and was the first example using a Reactive Control approach (well described in Chap. 14 of Matarić (2007)). The “Video 5.6” described the main steps for a Reactive Control approach:

1. Determine the “mutually exclusive conditions” that the bot would encounter during its operation in the world.
2. Match up each of those conditions with “appropriate” action(s). Create a table of matching conditions and actions to keep track of one’s current thinking about the problem to be solved.
3. From this “Condition|Action” table, generate a matching “Input|Output” or “Sensor|Actuator” table that would correspond to the actual sensors and actuators used in the bot design (see Fig. 5.15a, b).
4. Next translate the “Sensor|Actuator” table into coding facilities that were provided with the chosen software development environment (see examples in Figs. 5.3, 5.4 and 5.6).
5. Ones can expect to revisit Steps 1 through 4 several times before achieving a satisfactory solution.

5.3 “Sequence Commander” Project

This project used the CarBot described in Fig. 5.7 and it was a 4-WD adaptation of the 2-WD Sequence Racer robot belonging to the BIOLOID STEM system. The sample code “SequenceCommander_CM-510-530.tsk” was designed to work on

CM-510/530 controllers, while the other version “SequenceCommander_CM-5.tsk” was made for a CM-5 controller using the Integrated Sensor AX-S1 (ID= 100).

The “Sequence Commander” robot operated in two distinct phases:

1. This robot started out in a “Learning” mode where it would wait for the user to press the Up-Down-Left-Right buttons on the controller, in any combination or sequence, but up to five buttons only. As soon as the user pressed the sixth button, the robot would sound out an “error” signal, clear all previous inputs and restart on its learning phase.
2. Once the user had entered a “legal” sequence of button presses (≤ 5 buttons), the user would press on the “START” button to make the robot shift into its “Replay” mode. The robot would then move in the directions as recorded in the sequence made by the user during the learning phase. Each forward or backward step would last 1 s, while the left and right turn maneuvers would last only 0.5 s.

There were some program design features needing to be discussed further:

- (a) Each of the four AX-12 motors was provided with its own speed parameter (Speed1 through Speed4). This feature was provided so that the user could fine-tune these parameters in case the robot did not move “straight enough” when commanded to do so. This problem may come from robot construction misalignments or from non-uniform performance among the 4 motors. In practice, it may even come from the non-uniform properties of the surface that the robot would be running on, but that issue would be beyond the user’s control.
- (b) The “Button_Standby” function (see Fig. 5.16) was very short, but it fulfilled an important task of adjusting the extremely fast processing speed of the controller to the rather slow speed of human motion (when the user pressed and released a given button).

Before the execution of this function (essentially statement 131), the controller had already determined and saved the value of the button just pressed by the user into a parameter named “Button” (statement 34). However, the user’s finger might still be hovering over the pressed button as human reaction times are at best in tenths of a second, while the controller can execute its commands in microseconds. This “WAIT WHILE” construct essentially halted the controller’s progress as long as it detected that some button(s) were still being pushed. Please note that we had to use “double negation” to achieve this goal, as “NOT (No Button Was Pressed)” was equivalent to “Some Buttons Were Being Pressed”. Please also note that it would be “computationally inefficient” to use a WAIT WHILE condition that

Fig. 5.16 Function to help with “debouncing” button presses

```

128 FUNCTION Button_Standby
129 {
130     // Waiting for user to release buttons
131     WAIT WHILE ( 🚦 Button != 🚦 -- )
132 }
```

involved “OR” operators and the status of each of the U-D-L-R buttons as shown below, although it would be logically equivalent.

```
WAIT WHILE ( (Button == U) || (Button == R) || (Button == D) || (Button == L) )
```

The video file “Video 5.7” described the code in more details and contained webcam recordings of this robot in action.

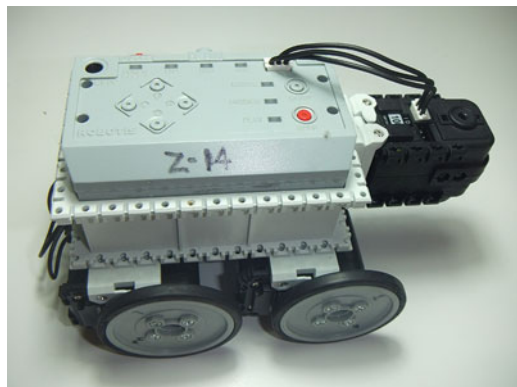
5.4 “Smart Avider” Project

This project used a CM-5 CarBot described in Fig. 5.17 and it used an AX-S1 for detecting obstacles in front of it.

True to its namesake of “Integrated Sensor”, the AX-S1 has many functions as shown in Fig. 5.18:

1. Active NIR sensing via “IR Fire Data” [0–255] (Addresses 26–28 for specific Left-Center-Right sensors). When these values are larger than a threshold value set at Address 52 (defaulting to 32), a 3-bit (RCL) flag would be set at Address 32.
2. Passive NIR sensing via “Light Data” [0–255] (Addresses 29–31 for specific Left-Center-Right sensors). Similarly as for active mode, when these values are larger than a threshold value set at Address 53, a 3-bit (RCL) flag is set at Address 33.
3. Detecting sound claps via parameters at Addresses 35–37.
4. Generating sound via buzzer (addresses 40 and 41).
5. Threshold settings for Active and Passive NIR sensing modes (addresses 52 and 53 respectively). When Address 52 is set to 0, the Active NIR sensors switch to a short-range mode and they can only detect objects within 12 cm of each sensor’s opening. But if Address 52 is set to a non-zero number, the Active NIR sensors go to a long-range mode whereas they can detect objects up to 37 cm away from their openings.

Fig. 5.17 CM-5 CarBot used



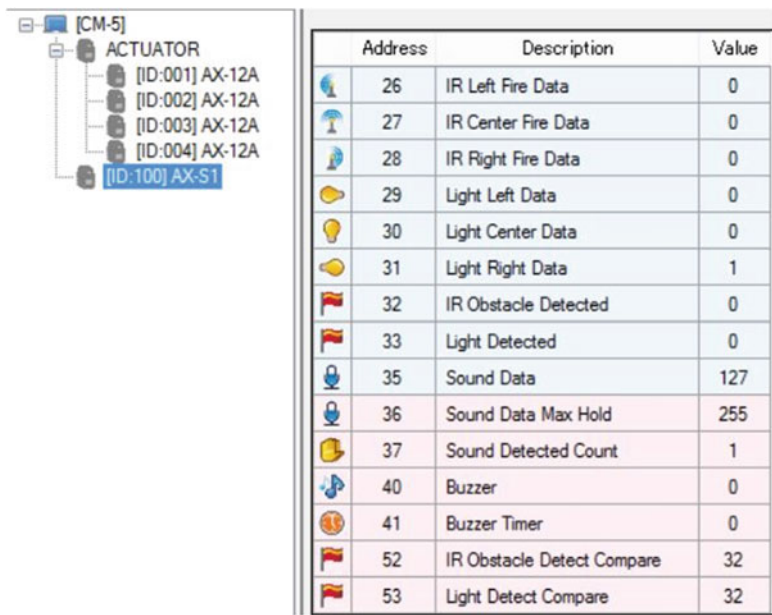


Fig. 5.18 MANAGER’s display for a CM-5 with AX-S1

In this section, the goal was to illustrate two control approaches (Reactive and Behavior-Based) on the same physical CarBot (Fig. 5.7). The Reactive Control approach yielded a solution represented by the TASK program “ObstacleDetectionCar.task”, while the Behavior-Based Control approach yielded “SmartAvoider.task”.

As previously discussed in Sect. 5.2.2, a Reactive Control approach required the generation of “mutually-exclusive” conditions for the robot to monitor along with specific and appropriate robot action(s) to be triggered when each of those conditions became true. Figure 5.19 was the Condition-Action table that was generated upon analysis of the “object-detection” capabilities of the AX-S1 (in NIR long-range detection mode).

The translation of this Condition-Action table into a TASK program yielded the code fragment as shown in Fig. 5.20 where the “Object Detected” Flag was used to represent the eight possible conditions to be monitored. Please note that a Reactive Control approach usually lead into an IF-ELSE-IF logical structure. The video clip “Video 5.8” had more detailed explanations of the program design and webcam views of the robot performance in avoiding obstacles.

Upon viewing the “Video 5.8”, the reader would notice that the behavior for dealing with a front obstacle was not very satisfactory as it kept on repeating seemingly un-intelligent actions. A possible remedy was to add a slight left or right turn to the MovFrd function (this is left to the reader to do as an exercise).

Reactive Control Approach	
1.	No obstacle detected >> Forward
2.	Obstacle in front >> Backward
3.	Obstacle on left >> Right
4.	Obstacle on right >> Left
5.	Obstacles on left & right >> Forward
6.	Obstacles on left & front >> Right
7.	Obstacles on right & front >> Left
8.	Obstacles in 3 directions >> Stop

Fig. 5.19 Condition-Action Table for CM-5/AX-S1 CarBot

16	Direct = ID[100]: Object Detected
17	IF (Direct == 0000 0000 0000 0000)
18	CALL MovFrd
19	ELSE IF (Direct == 0000 0000 0000 0010)
20	CALL MovBrd
21	ELSE IF (Direct == 0000 0000 0000 0001)
22	CALL MovR
23	ELSE IF (Direct == 0000 0000 0000 0100)
24	CALL MovL
25	ELSE IF (Direct == 0000 0000 0000 0101)
26	CALL MovFrd
27	ELSE IF (Direct == 0000 0000 0000 0011)
28	CALL MovR
29	ELSE IF (Direct == 0000 0000 0000 0110)
30	CALL MovL
31	ELSE IF (Direct == 0000 0000 0000 0111)
32	CALL Stop

Fig. 5.20 IF-ELSE-IF structure corresponding to Condition-Action Table of Fig. 5.19

Matarić (2007) presented a compact introduction to the Behavior-Based Control (BBC) approach (Chap. 16) but the seminal work in this area was from Arkin (1998). Matarić described BBC as “the use of behaviors as modules for control” and that

```

15  ENDLESS LOOP
16  {
17      IF ( ID[100]: IR Left > 200 || ID[100]: IR Right > 200 )
18          CALL Avoiding_side
19      IF ( ID[100]: IR Center >= 252 )
20          CALL Escape
21          CALL Go_forward
22  }

```

Fig. 5.21 Code fragment corresponding to BBC as applied to SmartAvoider

“behaviors achieve and/or maintain particular goals”. When translated to the SmartAvoider solution, there were three behaviors to emulate:

1. How to go forward.
2. How to avoid obstacles that come in from the side.
3. How to escape from dead-ends.

Figure 5.21 displayed a code fragment representing the main logic of the “SmartAvoider.tsk” program whereas the robot would go forward as its default behavior, but it would also additionally act appropriately to avoid side obstacles and to escape from cul-de-sacs.

The reader would surely appreciate the coding differences between Figs. 5.20 and 5.21 (i.e. “IF-ELSE-IF” vs. “parallel IFs” with a default function call Go_forward). The video clip “Video 5.9” would present more explanations about the coding of those three behaviors and about additional programming decisions taken to improve NIR sensors use. This video clip closed by showing the performance of this “unaltered” SmartAvoider robot as it tried to navigate through a maze, although it was not originally designed for this purpose. This was to illustrate the concept of “emerging behavior” (which was not completely successful in this example as the robot struggled very hard to get out of the cul-de-sac). The interested reader can also visit YouTube for other performances by the same CM-5 CarBot on a different maze (<http://www.youtube.com/watch?v=tR0dWVppFFM> and <http://www.youtube.com/watch?v=iFgVpc5U7Rg>).

5.5 “Line Tracer” Project

This project came from the BIOLOID STEM STANDARD kit (CM-530) and used two AX-12W for the motorized wheels and the IR Sensor Array (IRSA) to find its way on a very complex line track (see Fig. 5.22).

It was well designed mechanically to make it nimble in maneuvers and it also illustrated the appropriate uses of timed turns and conditional turns.

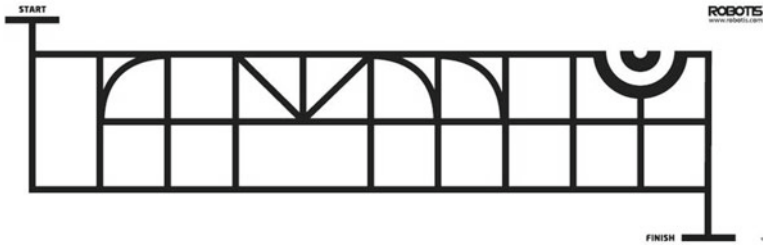
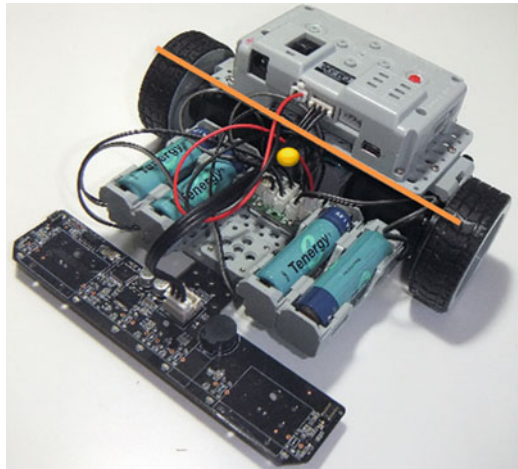


Fig. 5.22 BIOLOID STEM Line Track

Fig. 5.23 Line Tracer's
CG and wheel contact line



5.5.1 Mechanical Design Features

Some well thought out mechanical design features needed to be noticed:

1. The batteries and AX-12Ws (the heaviest components) were laid out such that the robot's center of gravity was just slightly ahead of the wheels contact line (Fig. 5.23). This helped the maneuverability of the Line Tracer.
2. Figure 5.24 showed the subtle ~1 mm clearance in the back, while the front weight rested on the two big LEDs of the IRSA. Thus this robot has 4 points of contact with respect to the travel surface and the front LEDs offered the minimum friction possible.

As a design exercise, the author adapted the BIOLOID STEM Avoider robot to also have a line-tracing ability using the IRSA (see Fig. 5.25). However as its CG was shifted more forward as compared with the Line Tracer's, this modified Avoider was found to be not as maneuverable as it tended to drag on the "larger" metallic hemispheroidal support piece when a turn was executed.

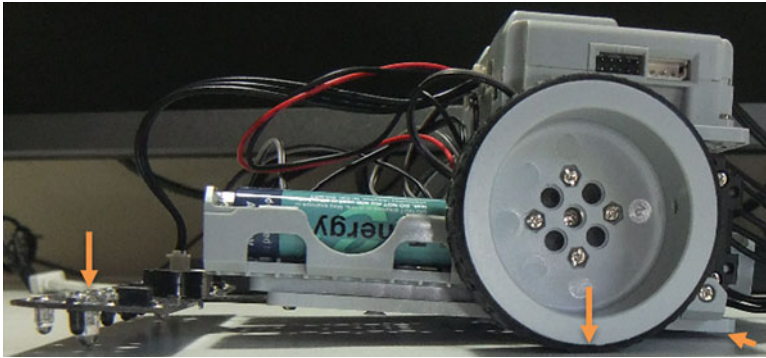


Fig. 5.24 Line Tracer’s points of contact with the travel surface

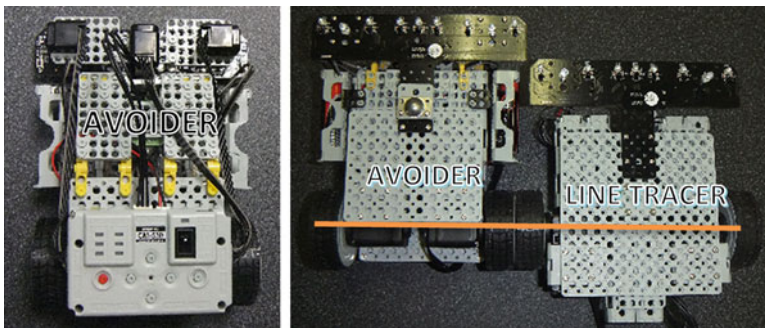


Fig. 5.25 Line Tracer’s points of contact with the travel surface

5.5.2 IR Array Sensor (IRSA)

The IRSA had seven NIR LED sets to detect changes in brightness, i.e. whether it was on the black track or not. The LED sets were positioned to match the IRSA with the width of the black track and also for the detection of the circular and diagonal branches (Fig. 5.26).

The video file “Video 5.10” demonstrated how to set up threshold values for each NIR LED set to result in a proper setting of the IRSA’s IR Obstacle Detected flag (a 7-bit parameter) which was an important parameter used in programming the maneuvers needed by the robot to navigate a user-defined path within the line track shown in Fig. 5.26.

5.5.3 Programming Maneuvers for Line Tracer

The TSK programs used in this section were adapted from the original TSK codes provided by ROBOTIS at their web site (http://support.robotis.com/en/product/bioloid/stemkit/download/bio_stem_standard_apps.htm) for their robot named “LineFollower”.

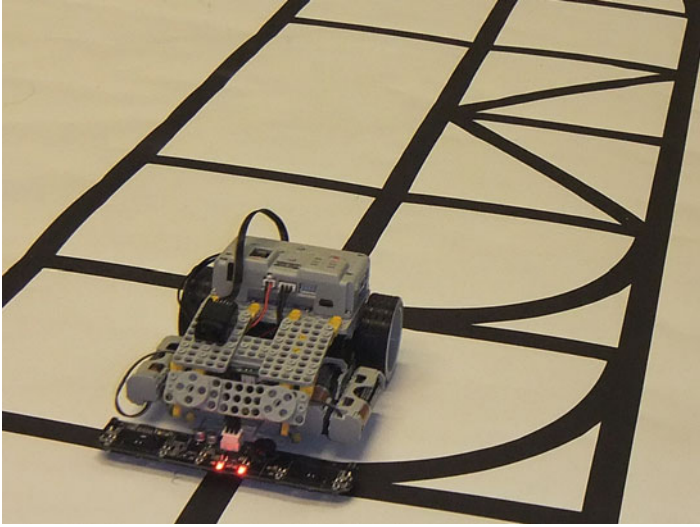


Fig. 5.26 Avoider robot on line track

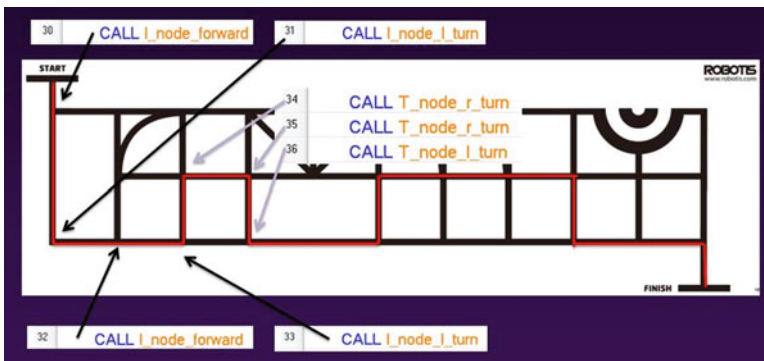


Fig. 5.27 Path (gray) taken by robot via LineTracer1.tsk

The “LineTracer1.tsk” program took this robot on a path that had only straight line runs and 90° turns (left and right). Figure 5.27 showed this path (in gray) and the corresponding function calls for the first seven maneuvers. The “Video 5.11” video file explained the programming aspects in more details.

The “LineTracer2.tsk” illustrated how the robot could detect and handle diagonal and curve “nodes”—see Fig. 5.28 and video file “Video 5.12” for more programming explanations.

ROBOTIS also provided a WMV video file of LineTracer2 program in action at this web site (http://www.robotis.com/video/BIO_STEM_LineFollower.wmv).

In “LineTracer3.tsk”, two tasks/missions were added to the “LineTracer2” code using light and audio activations (Fig. 5.29).



Fig. 5.30 Different versions of the RC-100

(any RC-100 version), or with BT-100 (versions A and B) or with BT-210 (version B only) to use BlueTooth protocols (http://support.robotis.com/en/product/auxdevice/communication_main.htm).

- For wireless communications and controls from a Personal Computer, the user has two options:
 - (a) If the user's PC has BlueTooth (BT) capabilities, then use BT-110 or BT-210 on the robots, and just access the appropriate COM ports created by the PC's BT services upon pairing and connection. Using BT protocols would allow TASK to perform "program download" during editing time and "remote control" at run time.
 - (b) If the user's PC has no BT capabilities, then the user needs to use a combination of USB2DYNAMIXEL-ZIG2SERIAL modules and also ZIG-100 or BT-100 modules for the protocol needed. Option (b) is obviously more expensive than option (a), but the USB2DYNAMIXEL and ZIG2SERIAL can perform other functions such as direct access to actuators or changing the baud rates. BT is potentially faster than ZigBee, but my personal experiences so far have shown that ZigBee is more reliable than BT. Also the user has to consider whether "one-to-one" BT connections are enough or that he or she would need broadcast-type communications in the future.

Depending on the controller type used, the TASK tool provides up to six communications related parameters (see Fig. 5.31):

1. "Remocon Data Received" is a logical flag which is set to TRUE when the controller received a new message, otherwise it is set to FALSE. This parameter is available on all controllers (CM-5, CM-510, CM-530 and OpenCM-9.04).
2. "Remocon RXD" is a 16-bit parameter corresponding to the message received (available on all controllers).
3. "Remocon TXD" is used to send out a 16-bit message (available on all controllers). This procedure will be demonstrated in Chap. 8.
4. "My ID" corresponds to the ZigBee ID of either the ZIG-100 or ZIG-110 module used on the robot being programmed. Currently, it is available on CM-5XX controllers only (i.e. Firmware 1.0), even though the author knows that ZigBee works with the OpenCM-9.04 system (i.e. Firmware 2.0), so maybe it will be supported in a future release of TASK?

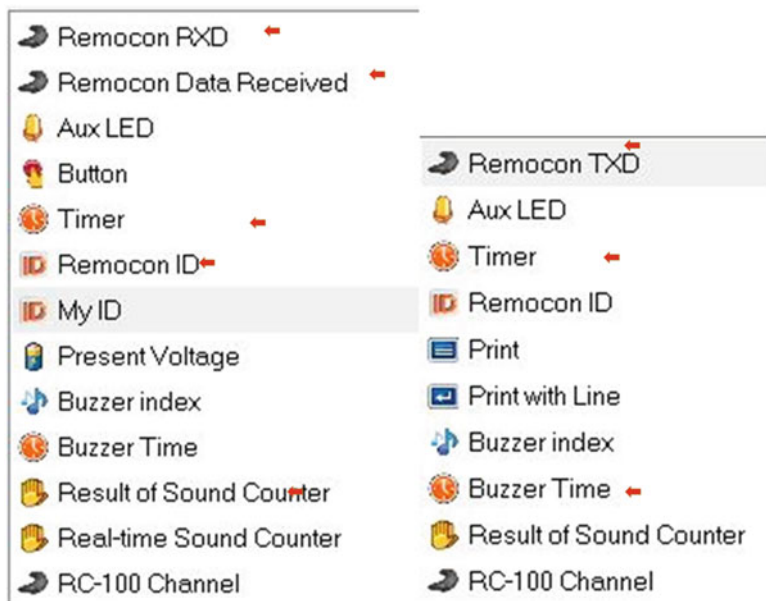


Fig. 5.31 Communication Parameters available in TASK tool

5. “Remocon ID” corresponds to the ZigBee ID for the “other” ZIG-100 or ZIG-110 modules (on the “other” robot or on the RC-100 being used). Currently, it is implemented on Firmware 1.0 controllers only.
6. “RC-100 Channel” only pertains to the NIR communication option.

More information can be found at the ROBOTIS e-Manual web site ([http://support.robotis.com/en/software/roboplus/roboplus_task/programming/parameter/controller\(roboplus_task\).htm](http://support.robotis.com/en/software/roboplus/roboplus_task/programming/parameter/controller(roboplus_task).htm)).

In this section, only the use of “Remocon Data Received”, “Remocon RXD” and “Remocon TXD” would be demonstrated. “TestComm.tsk” was made for an OpenCM-9.04-C using 4 XL-320s (ID=[1, 2, 3, 4]). Figure 5.32 listed a code fragment to illustrate the basic algorithm:

- After initializing the XL-320s to wheel mode and setting their speeds to a value of 512 (CALL Initialize), the controller went into an endless loop whereas:
 - (a) It waited for new data to come in, i.e. stayed at Statement 7 as long as “Remocon Data Received” was FALSE.
 - (b) When this flag became TRUE, because new data just came in, it would save this data in parameter “Data” (Statement 8).
 - (c) Then it would go through a procedure to determine the digit that corresponded to the “thousand” position in “Data” (i.e. parameter “Thousands” in Statement 10). If “Thousands” is positive, it would turn Motor 1 as many times as the actual value found in “Thousands”. If not, it would flash on the LED of Motor 1 once for 0.5 s.

```

3      CALL Initialize
4
5      ENDLESS LOOP
6      {
7          WAIT WHILE ( Remocon Data Received == FALSE )
8          Data = Remocon RXD
9
10         Thousands = Data / 1000
11         Print = Thousands
12         ID = 1
13         IF ( Thousands > 0 )
14         {
15             LOOP FOR ( I = 1 ~ Thousands )
16             {
17                 CALL Motors
18             }
19         }
20         ELSE
21         {
22             CALL LED
23         }

```

Fig. 5.32 Algorithm to parse out the “Thousand” digit from “Data”

- (d) Then it would proceed similarly to determine and handle the digits corresponding to the “hundred” (Motor 2), “ten” (Motor 3) and “unit” (Motor 4) positions in “Data”.
- (e) At the end of the endless loop, the robot would send back, via “Remocon TXD”, to the PC the original “Data” that it received (Statement 75). The robot would also “Print Line” back to the PC this original “Data” (Statement 76). This was to show that the “Remocon TXD” command sent data in a different format, i.e. in a 6 byte packet that the “Program Output Monitor” window could not unpack and decode properly, thus it could only display “U” instead of the actual data. On the hand, because the “Print Line” command sent data in the clear, the “Program Output Monitor” could display the data properly (see video file Video 5.13).

The video file “Video 5.13” also explained more details for this program and showed how to use the “Zig2Serial Management” feature of the RoboPlus Manager tool to provide inputs from the PC to this program wirelessly via ZigBee (ZIG-100/110A) and Bluetooth (BT-110/210) devices.

The next three TASK programs were created to illustrate the application of RC concepts to a CarBot of a type as shown in Fig. 5.17:

1. “RC100_Carbot_1.tsk” demonstrated the use of the “&” (i.e. AND) operator to filter out unwanted user button pushes unless they were the Up-Down-Left-Right

buttons (Statement 18). It also showed the use of the IF-ELSE-IF structure to enable the activation of one and only one condition, in other words allowed the use of only one button on the RC-100 at any one time.

2. “RC-100_Carbot_2.tsk” showed the use of parallel IFs so as to enable the user to activate several buttons at the same time, for example “Up” and “Left” together to perform a left turn with a wider arc instead of a pivot turn when “Left” is used alone.
3. “RC-100_Carbot_3.tsk” built on “RC-100_Carbot_2.tsk” and added a new functionality by allowing the user to change the robot speed “on the fly” using the buttons “1-2-3-4”.

The video file “Video 5.14” had more details about these three TSK programs and showed how they performed at run time.

5.7 Review Questions for Chap. 5

1. What is the name of the sensor module in the Bioloid robotics system?
2. What is the name of the servo motor module in the Bioloid robotics system?
3. How many UART ports does the Bioloid Controller CM-5 provide?
4. What are the two ways that we can use the AX-12+ module for?
5. How many symbols are used in a hexadecimal numbering system?
6. What is the numerical range on decimal values that a 10-bit number can represent?
7. What is the UART_1 communication port used for on the CM-5?
8. What are the name and default speed (how many bits per second) for the communication protocol between the CM-5 and the various servo motor and sensor modules?
9. Please name two possible control architectures that can be used in robotics.
10. Which tool of the RoboPlus Suite to use when programming several servo motors in concert?
11. What is the type of communication protocol used between the PC and the typical CM-5XX controller.
12. Draw up the diagram for the World-Computer Interactions.
13. In the TASK tool, what are the two types of commands that can be used to set the rotation direction and speed for the AX-12 module?
14. In the TASK tool, when should we use the “CUSTOM” option?
15. What is the actual time elapsed when the standard timer counter changed by 3 counts?
16. The Bioloid system uses a Full-Duplex communication protocol. (T-F)
17. Inside the TASK tool, which parameter should you set in order to change the range mode of the AX-S1, from long range to short range, and vice-versa?
18. Which parameter should we set in the Task tool to make the AX-S1 NIR sensors go into the Short Range detection mode? Parameter name and correct value to set?

19. What is the minimum time period between sound claps for the AX-S1 to discern them as distinct sound claps?
20. How long does it take before a sound count detected in the AX-S1 becomes accessible to a program running on the CM-5?
21. The NIR sensors on the AX-S1 modules can be set to be used in a passive mode. (T-F)
22. Which parameter should we set in the Task tool to make the AX-12+ modules go into the Wheel Mode? i.e. Address number and correct value to set?
23. What is the “duplex” type of the communication protocol used by the Bioloid system to communicate between different devices?
24. To control the accuracy of the Goal Position on an AX-12, we need to reduce its SLOPE value to the minimum value of 1. (T-F)
25. What is the purpose of an IF-ELSE-IF structure?
26. What is the purpose of a parallel IFs structure?
27. Which communication protocol does the Virtual RC-100 Controller use to communicate with the CM-5XX controller?
28. Describe procedure to ignore all inputs from the RC-100 except for buttons “2” and “3”.

5.8 Review Exercises for Chap. 5

1. To practice setting an AX-12 into Continuous Turn mode and using basic timed operations for it with the LOAD or COMPUTE commands, please use the enclosed programs “AX12ContinuousTurn_1.tsk” and “AX12ContinuousTurn_2.tsk”.
2. To practice reading from the DMS and NIR sensors, please use the enclosed program “DMS-NIR_sensing.tsk”.
3. To practice using the AX-S1 as a light sensor, please use the enclosed program “AX-S1-LightSensing.tsk”.
4. To practice using the AX-S1 as a NIR sensor in a direct mode, please use the enclosed program “AX-S1-IR-SensingDirect.tsk”. This program also shows how to get the AX-S1 into its long-range and short-range mode.
5. To practice using the AX-S1 as a NIR sensor using its Obstacle Detected Flag and its related Threshold setting, please use the enclosed program “AX-S1-IR-SensingFlag.tsk”. Please modify this program to make it work similarly but in its Light Sensing mode.
6. The enclosed program “CM5Music.tsk” demonstrates how to get the AX-S1 to play music using a CM-5. Please modify this program to make it work on a CM-530 controller.
7. Expand enclosed “Sequence Commander” code so that the bot can handle up to eight maneuvers in a given sequence.
8. Modify “Sequence Commander” code so that each left and right turn corresponds to a 90° turn. Does your solution work for all running surfaces? And at all battery levels? i.e. these “timed” maneuvers have their limitations and more flexible “sensor-based” approaches are used in the “Line Tracer” project in Sect. 5.5.

9. Please start from the sample program “RC100_Carbot_2.tsk” and implement the following features:
 - (a) If no button is pushed on the RC-100, the carbot should stop.
 - (b) Buttons U-D-L-R are still used in the usual manner.
 - (c) Buttons 1-2-3-4 will be used to adjust the nominal speed level of the carbot:
 - Button 1 will set the speed to 128 (initial default speed).
 - Button 2 will set the speed to 256.
 - Button 3 will set the speed to 512.
 - Button 4 will set the speed to 1023.
 - The user should have to push any of the buttons (1 through 4) only ONCE to set the “new” speed (i.e. the user SHOULD NOT have to keep pushing down on the “speed” buttons to affect a speed change).
 - Please make sure that the user can change the speed setting (1, 2, 3, 4) while performing a car maneuver (U,D,L,R) AT THE SAME TIME.
 - Remember that you can use the Virtual RC-100 (i.e. using Mouse click or keyboard from inside the Task Tool—the View Print of Program sub-window to be exact) to test out your solution before using the Physical RC-100.
10. Practice in combining Remote Control and Autonomous Behavior programming in one application:
 - Simulation of a tail-gating situation between two carbots (both under remote control by the RC-100’s), when the front carbot suddenly stops (an equivalent static frontal obstacle could also be used).
 - Rear carbot using its NIR sensors will trigger an autonomous response (i.e. ignoring commands from RC-100) to help it avoid colliding into front car).
 - The Spring 2013 solutions from the students at National Taiwan University can be viewed at this link https://www.youtube.com/playlist?list=PLVHBjRDK0kAJA_GC3UMreuKVeWnuASw7P&feature=view_all.

References

- Arkin RC (1998) Behavior-based robotics. The MIT Press, Cambridge
- Campion G, Chung W (2008) Wheeled robots. In: Siciliano B, Khatib O (eds) Springer handbook of robotics. Springer, Heidelberg, pp 391–410
- Cook G (2011) Mobile robots. Wiley, Hoboken
- Dudek G, Jenkin M (2010) Computational principles of mobile robotics. Cambridge University Press, Cambridge
- Matarić MJ (2007) The robotics primer. The MIT Press, Cambridge
- Nourbakhsh IR (2013) Robot futures. The MIT Press, Cambridge
- Winfield A (2012) Robotics. Oxford University Press, Oxford

Chapter 6

Actuator Position Control Basics

In Chap. 3, the hardware characteristics for representative actuators such as the AX-12 and MX-28 were described and the key information was on the rotational encoders that were used for the AX-12/18 (a variable potentiometer, restricted to a 300° range) and for the MX-28/64 (a magnetic field detector allowing a full 360° range). In this chapter, the programming aspects for controlling such actuators via the TASK and MOTION EDITOR (V.1) tools are described. The capabilities of the newer R+MOTION (V.2.1) tool will be discussed in Chap. 11 using the DARWIN-MINI and the XL-320.

This chapter's main topics are listed below:

- How does an actuator reach and maintain a given position via TASK?
- Concept of Motion Page in MOTION EDITOR (V.1) and application to a biped robot.
- Walking robots—Forms and Functions: BUGFIGHTER, HEXAPOD, DROID, GERWALK, BIPED and HUMANOID.

For more advanced topics on legged and humanoid robots, the reader is referred to Kajita and Espiau (2008), Kemp et al. (2008), Chevallereau et al. (2009), Abdel-Malek and Arora (2013) and Kajita et al. (2014).

6.1 AX-12/18 Position Control with TASK

In Position Control mode, the AX-12/18 was restricted to a range of 300°, with 0° (i.e. Goal Position=0) located in the lower right corner of the actuator front view (see Fig. 6.1), and 300° (i.e. Goal Position=1023, i.e. a 10 bit parameter) located in its lower left corner (Fig. 6.1).

Electronic supplementary material: The online version of this chapter (doi:[10.1007/978-3-319-20418-5_6](https://doi.org/10.1007/978-3-319-20418-5_6)) contains supplementary material, which is available to authorized users.

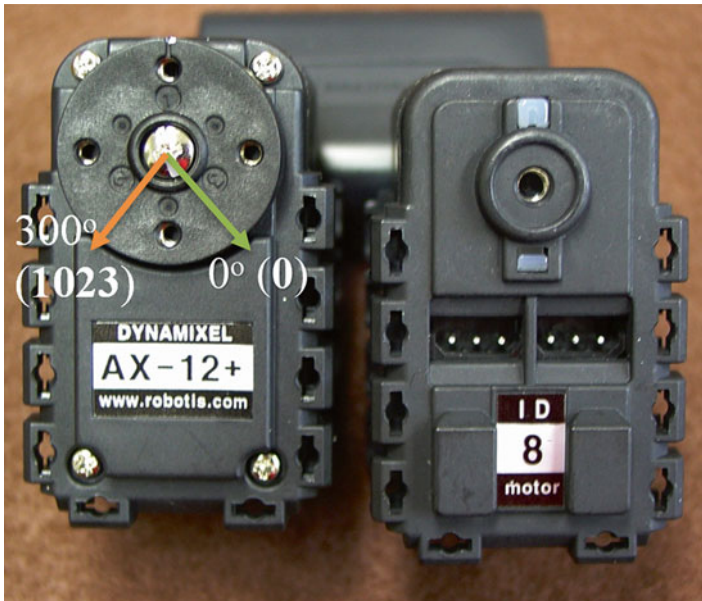


Fig. 6.1 Front and back views of AX-12

```
FUNCTION Initialize_motor
{
    // Set_Position Control mode to all servos
    ID[1]: ADDR[8(w)] = 1023
    ID[2]: ADDR[8(w)] = 1023
    ID[3]: ADDR[8(w)] = 1023
    ID[4]: ADDR[8(w)] = 1023
    ID[5]: ADDR[8(w)] = 1023
    ID[6]: ADDR[8(w)] = 1023
    ID[7]: ADDR[8(w)] = 1023
    RETURN
}
```

Fig. 6.2 TASK function to initialize AX-12s to Position Control mode

First, ones must take care in setting all actuators to the Position Control mode before programming them in TASK for this mode. This action could be performed inside the RoboPlus Manager tool by setting the “CCW Angle Limit” parameter to a value of 1023 or by using a LOAD command inside a TSK program with a CUSTOM WRITE of a WORD-sized value of 1023 into Address 8 of appropriate Dynamixel IDs (preferred by author, see Fig. 6.2).

The Control Table for all operational parameters for the AX-12 (as an example) could be accessed at this link http://support.robotis.com/en/product/dynamixel/ax_series/dxl_ax_actuator.htm. Please note that most parameters had Read/Write properties, but some were set to Read-Only. The user also should be aware that the **name** for a particular parameter as shown in this **Control Table** may not match with the **name** for the same parameter as shown in the **TASK** program pop-up window, so it is highly recommended for users to rely on the **address** of such parameter instead. For example, Address 34 in the Control Table was named “Torque Limit” (allowable range [0–1023], but the same parameter in TASK tool was named “Goal Torque” (see Fig. 6.3). **This book would only use the parameter names as provided in the Control Table web links (hopefully by the time this book goes to print, the typos inside the TASK tool would have been corrected).**

Figure 6.3 also listed often-used parameters:

1. “Goal Position” [0–1023] stood for the location where the user wanted the actuator to go next.
2. The CW/CCW “Margin” and “Slope” parameters worked in concert to provide the “mechanical behavior” of the actuator when it traveled from a “Present Position” to the already set “Goal Position” (more details later in this chapter).
3. “Torque Limit” [0–1023] denoted the maximum load (electrical current) allowed on the actuator. In practical terms, when the “Torque Limit” was set to 0 for any actuator, the user could rotate such actuator by hand (as no electrical current was

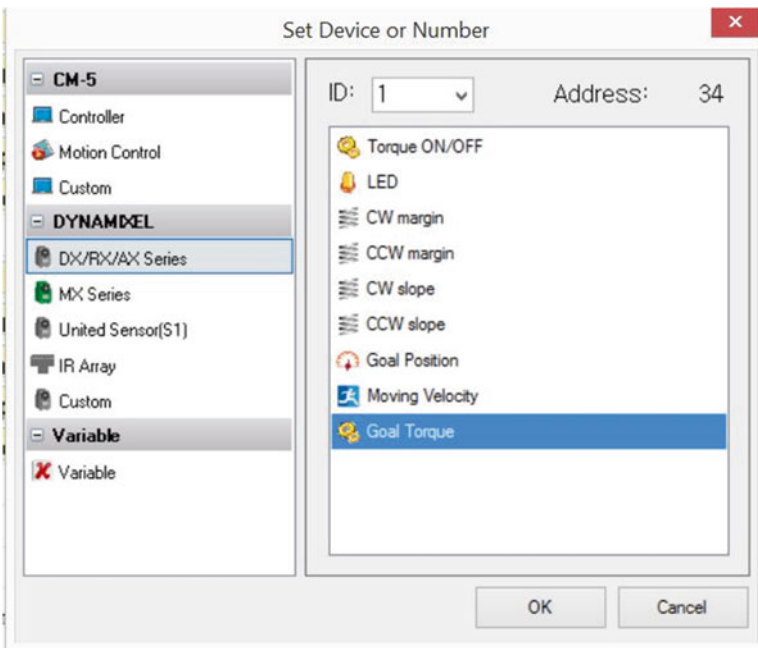


Fig. 6.3 Example of a parameter naming error in TASK tool

allowed into the motor, thus no resisting torque to maintain a given “Goal Position”). This effect would be demonstrated in Chap. 7 for the implementation of a Bilateral Control of 2 GERWALKS robots. “Torque Limit” worked in concert with “Moving Velocity” to provide the actual (final) rotational speed of the actuator.

4. “Moving Velocity” should be considered as a relative setting [0–1023], whereas 0 meant to stop the actuator, and 1023 meant the maximum speed allowable by the current setting of “Torque Limit”. For example, a setting combination of “Torque Limit” (=512) and “Moving Velocity” (=1023) would only result in half the rotational speed that the actuator was capable of doing physically. Users also would not need to set the rotation directions—CW or CCW when actuators were in Position Control mode (in contrast to when they were in wheel mode for Chap. 5).

For actuators, there was another important parameter called “Punch” that users could adjust by using a CUSTOM WRITE of a WORD-sized value (default=32) into Address 48 of appropriate Dynamixel IDs. It corresponded to the minimum electrical current used to drive the motor from a full-stop, i.e. to overcome static friction and inertia (parameter E in Fig. 6.5).

Another issue to be aware of when using TASK was that just because a parameter showed up in the pop-up menu did not mean that it really existed and was operational for the actuator being used for the user’s particular robot. In Fig. 6.4, the “Goal Torque” parameter (address 71) was really operational for the MX-64 actuator only and did not exist at all on the Control Table of the MX-28, please see links to both Control Tables below:

- http://support.robotis.com/en/product/dynamixel/mx_series/mx-28.htm
- http://support.robotis.com/en/product/dynamixel/mx_series/mx-64.htm

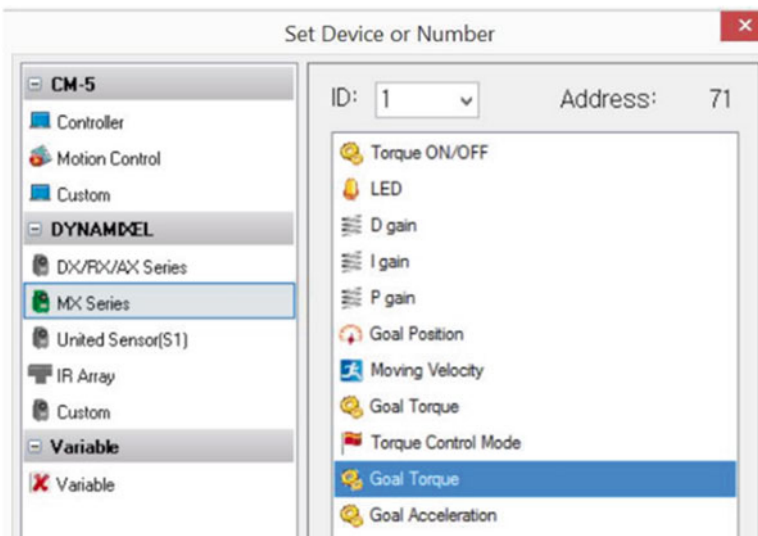


Fig. 6.4 “Goal Torque” (address 71) operational only for MX-64 and not for MX-28

3. AX-12-MoveMargin.tsk
4. AX-12-MoveSmooth.tsk
5. AX-12MonitorPositionSpeed.tsk

6.2 Using Motion Editor (V.1)

Currently, ROBOTIS is supporting two motion-editing tools, one bundled with the RoboPlus suite (Motion V.1) and the other, self-standing, called R+Motion (currently at V.2.1.1). Motion V.1 had been around since 2009 and is currently at v1.0.29.0. Motion V.1 still supports some CM-5 based robots (BIOLOID COMPREHENSIVE kit), so it may still be around for a few years but its days are numbered. R+Motion V.2 only supports the BIOLOID PREMIUM kit and later kits, such as STEM, SMART and ROBOTIS-MINI and this tool will be discussed in more details in Chap. 11.

As pointed out in Sect. 4.4, conceptually these Motion tools were based on cartoons (or frames) animation which was essentially the re-playing of a sequence of poses of the considered character (robot) within a time line, with slight variations in each pose to create the illusion (perception) of motion. The key idea is to use MOTION tools to create motion **data sets** (saved as *.MTN files) while using the TASK tool to create the **logical flow** among these motion data sets (saved as *.TSK files). In other words, the two tools, MOTION and TASK, have to work together in order to create robot moves that are both mechanically realizable and logically coherent.

The MOTION V.1 tool is quite a sophisticated tool and the user can purchase a copy of the User's Guide at http://www.robotis-shop-en.com/?act=shop_en.goods_view&GS=1486&GC=GD080400. This manual is highly recommended as it provides extensive procedures on how to use specific features of the tool which won't be repeated in this book. Instead, this book will strive to provide an integrative description of key concepts and practical demonstrations of selected procedures via video recordings.

6.2.1 Characteristics of a Motion Page in Motion V.1

Motion V.1 was designed to operate with an actual robot connected via a serial communication port in real-time to the PC running this tool. Figure 6.6 showed the complete window for Motion V.1.0.29.0 when started up on a PC. It contained two main panels:

1. The right panel was designed to do pose editing, i.e. to interact directly with the robot by turning selected actuators on and off, and thus enabling the setting of the robot into various "poses" which were then saved as sequential "steps" on the left panel.

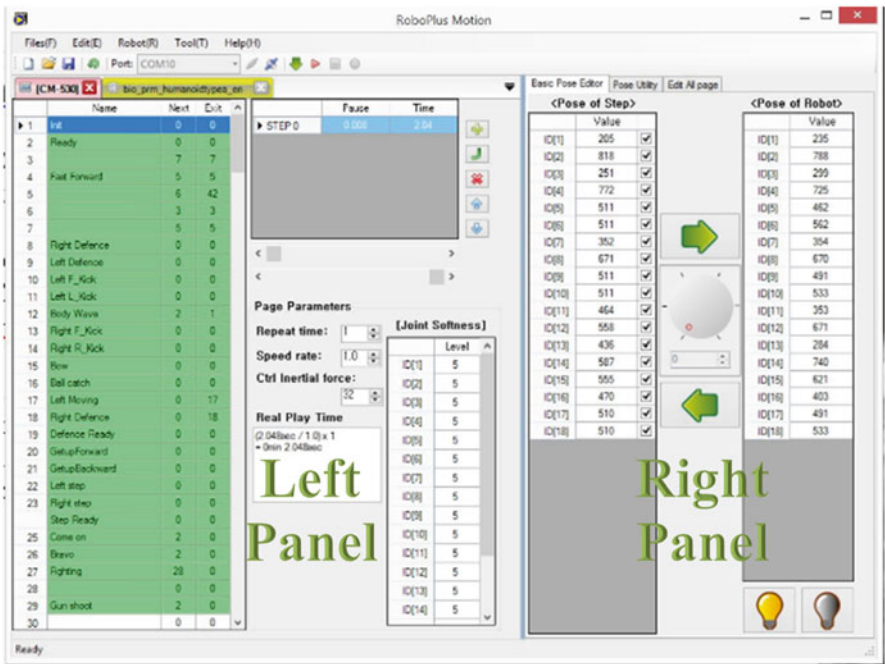


Fig. 6.6 Motion V. 1 tool (full view)

2. The left panel was designed to allow the grouping of these “sequential steps” into “motion pages” which could be left self-standing or further linked together for more extensive robot moves.

Conceptually, the user would need to consider the Motion Pages, located on the left edge of the Left Panel, as “macro” lines of code, and as such, they could be copied and pasted somewhere else for further editing. They could be renamed as needed, or named if a blank page was created.

Each Motion Page could have up to seven steps (i.e. robot poses) labeled from Step 0 to Step 6. A **Pose** was essentially a data set of “n” actuator positions [0–1023] where “n” was the number of actuators used for generating robot moves (thus not all actuators making up the robot needed to be listed here). A **Step** was then a Pose along with a play **Time** period (0.072–2.04 s) allocated to the robot for it to reach this Pose, from whatever its actuator positions were at the start of this Step. A **Pause** time (0.000–2.04 s) was optional and specifies the time period for the controller to wait out (doing nothing) before it can start on the next Step.

Each Motion Page could be connected to another Motion Page by specifying that Page Number in the “**Next**” column (see Left Panel), i.e. after all the steps in the current Motion Page were executed, the controller would play the “Next” Page Number as specified in its own “Next” cell value. A “0” in this “Next” cell meant for the controller to stay at the same Motion Page when the last Step of the current

Motion Page was finished. For example, as shown in Fig. 6.6, let’s trace the route that the robot would take if Motion Page 4 (Fast Forward) was initiated. After Page 4 was done, it would go to Page 5, then Page 6, then Page 3, then Page 7, then back to Page 5, i.e. the robot would “Fast Forward” continuously until its power is turned off or the controller gets reset.

It was also possible that a command to stop a Motion Page needed to be executed while the robot was still going through a particular Motion Page, and as a consequence of this command, the robot could be put into an unstable situation/pose. To alleviate this problem, the user could specify a defined Motion Page as an “Exit” Page where the robot could go to, after an emergency stop command had been issued. For example, if the robot was doing Page 5 when it was ordered to stop, it would “exit” to Page 42 which would lead to Page 43 for a final stop, setting the robot into a ready position (Fig. 6.7).

Found in the middle of the Left Panel, the “Page Parameters” section (Fig. 6.8) could be used to specify four parameters for each Motion Page, independently of each other:

Fig. 6.7 Exit Page 42, eventually ending at Page 43

42	F_E_R	43	43
▶ 43		0	0

Fig. 6.8 Page Parameters that affect the final execution of Motion Pages

Page Parameters

Repeat time: 1

Speed rate: 1.0

Ctrl Inertial force: 32

Real Play Time

(3.984sec / 1.0) x 1
= 0min 3.984sec

[Joint Softness]

	Level
ID[1]	5
ID[2]	5
ID[3]	5
ID[4]	5
ID[5]	5
ID[6]	5
ID[7]	5
ID[8]	5
ID[9]	5
ID[10]	5
ID[11]	5
ID[12]	5
ID[13]	5
ID[14]	5

- “Repeat Time” and “Speed Rate” were rather self-explanatory parameters (see the formula used in the Real Play Time box).
- The “Control Inertial Force” (CIF) parameter (0–127 range, default value=32) represented the level of acceleration and deceleration used to move the robot **from Pose to Pose**. It was inversely proportional to the actual acceleration/deceleration level. Thus a small CIF translated to higher speed increases from the starting Pose and also higher speed decreases when near the ending Pose (like for a “punch”), so outwardly the robot would move fast but might shake and get unstable. Conversely, a large CIF would be equivalent to a “caress” then.
- The “Joint Softness” (JS) parameter (1–7 range, default value is 5) was related to the compliance (Margin and Slope) settings for the actuator, so it applied only when near the starting and ending Poses of a Motion Step. When JS was large, resulting movements would be smooth but would not be suitable for legs that needed much support. So the robot could collapse at the Ready Pose already. A small JS would provide steadiness of motion but resulting movements might look too rigid.

Motion V.1 also offered a Calibration tool which could be used to define actuator **offsets** that were particular to a given robot, as no robot could really be built exactly (in a mechanical sense) like another of the same model. However, we would expect them to be able to use the same Motion files and Task programs. This tool will be demonstrated in Sect. 6.2.2.

6.2.2 Application to a GERWALK Robot

To illustrate the main concepts described in Sect. 6.2.1, a CM-510 GERWALK robot (Fig. 6.9) was used to create the following videos:

- “Video 6.2” demonstrated the concept of a Calibration Pose and showed how to use the Calibration sub-tool and to do basic motion editing.
- “Video 6.3” used the sample Motion and Task files provided by ROBOTIS to illustrate the planning of motion design inside MOTION V.1 and the integration of Motion Pages with the logic flow design provided inside TASK.

6.3 Form and Function of Walking Robots

In this section, the goal was to illustrate how Form influences Function in the design and operation of walking robots, going from a design with only 2 actuators (BugFighter) to a Humanoid design with 18 actuators.

The author could be wrong on this issue, but so far the author could not find any legged robot that could move around on only one actuator. Even the Planar One-Leg Hopper from MIT (c. 1982) needed two actuators (http://www.ai.mit.edu/projects/leglab/robots/2D_hopper/2D_hopper.html).

Fig. 6.9 A CM-510 GERWALK Robot in Calibration Pose

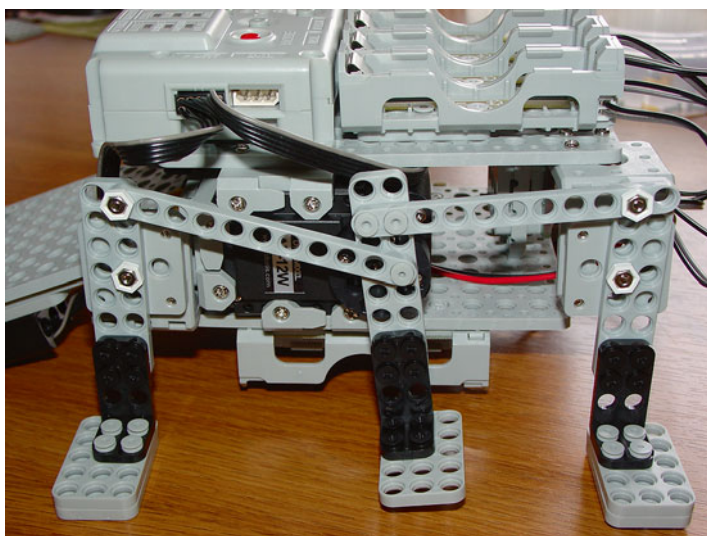
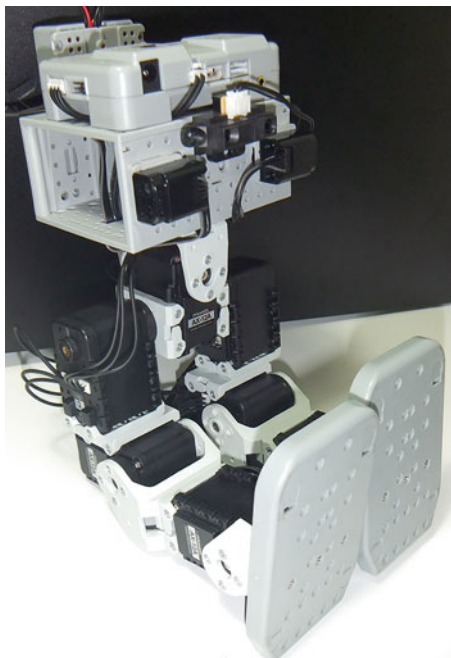


Fig. 6.10 Dual set of 4-bar linkages used for BugFighter

The BIOLOID STEM STANDARD kit offered a 2-servo BugFighter robot using two AX-12W in wheel mode. The walking motion of its six legs was based on the conversion of the rotary motion of the AX-12W into a back-and-forth motion for its legs by using a dual set of 4-bar linkages for each side (Fig. 6.10). Please note that

the 4-bar linkages worked in a vertical plane for this robot. Thanks to this mechanical conversion, the control scheme for this robot was the same for the various carbots used in Chap. 5. ROBOTIS has a video of this robot in action at http://www.robotis.com/video/BIO_STEM_BugFighter.wmv.

In the BIOLOID STEM EXPANSION kit, the Hexapod robot used three AX-12A in Position Control mode to achieve its walking ability: one actuator to shift its weight left and right using its middle leg in a seesaw motion (Fig. 6.11), and one actuator on each side to provide the forward-backward and steering motions. The hexapod also used a 4-bar linkage but as a parallel linkage between its front and back leg, and it worked in a horizontal plane instead (Fig. 6.12).

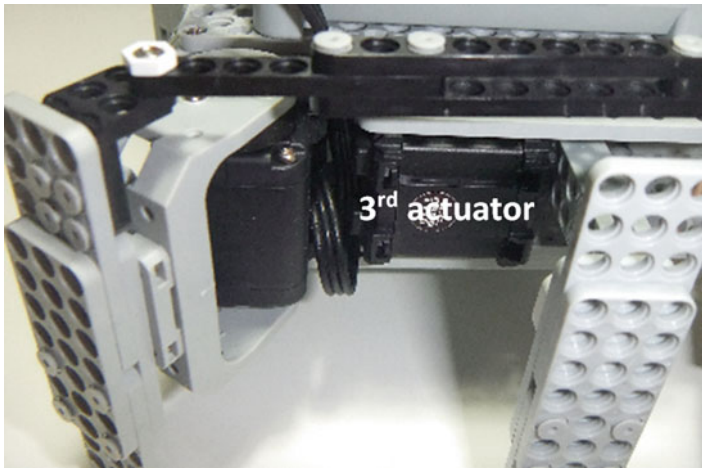


Fig. 6.11 ROBOTIS Hexapod middle leg (third actuator) is used to shift its weight *left* and *right*

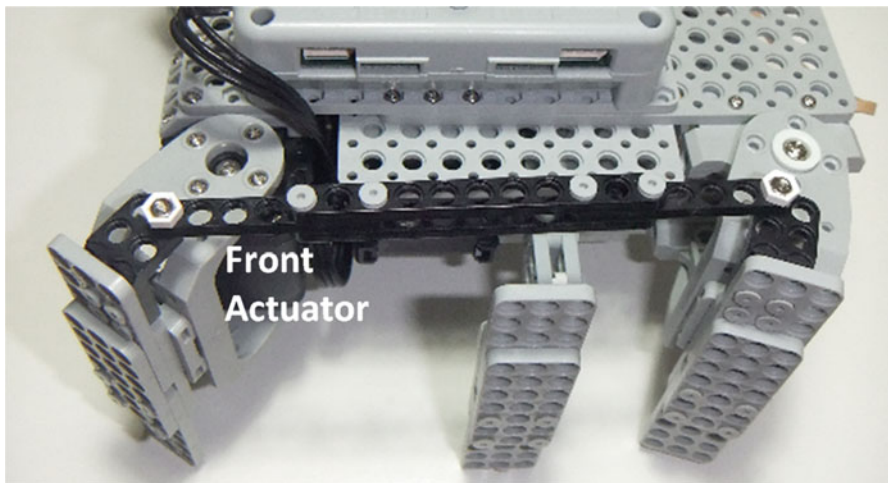
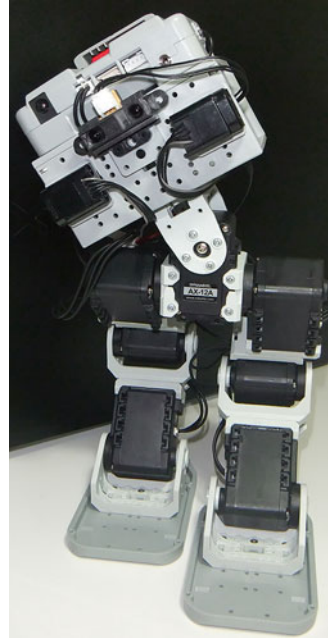


Fig. 6.12 ROBOTIS Hexapod front actuator's activation of a parallel 4-bar linkage to obtain its walking motion

Fig. 6.13 ROBOTIS
PREMIUM GERWALK



ROBOTIS has a video of its walking motion at http://www.robotis.com/video/BIOLOID_STEM_14.Hexapod.wmv.

TWITCH was also another 3-actuator Hexapod using a different design for its legs (http://forums.trossenrobotics.com/robots.php?project_id=7#ad-image-0).

There are great web resources for mechanical linkage design such as:

- <http://www.mekanizmalar.com/>.
- Kinematic Models for Design Digital Library from Cornell University (<http://kmoddl.library.cornell.edu/>).

Figure 6.13 showed a CM-510 version of the GERWALK using seven actuators and having a walking gait closer to the human solution. It swung its upper body left and right in order to shift its center of gravity towards the supporting leg before making a step with the other leg. As it had no hip joint, it had to use a “moon-walk” like maneuver to generate enough friction forces from the ground surface to allow it to turn sideways (see video at http://www.robotis.com/video/BIO_PRM_GerWalk.wmv).

Figure 6.14 showed a CM-510 version of the BIPED robot using eight actuators and having a walking gait even closer to the human solution. It did have ankle joints but it still had no hip joint, so it had to rotate its ankle and to use another “moon-walk” like solution to generate enough friction forces from the ground surface to turn sideways (see video at http://www.robotis.com/video/BIO_PRM_BipedWalkingRobot.wmv).

Fig. 6.14 ROBOTIS
BIPED



Fig. 6.15 ROBOTIS HUMANOID-A (lower body)

Figure 6.15 showed a CM-530 version of a Humanoid-A robot using 12 actuators for its lower body, i.e. it had hip and ankle joints and thus had a walking gait closer to a human gait (see “Video 6.4”).

The reader is also recommended to read and practice the materials found in Section 4-4 (Advanced Learning) of the Software Programming User’s Guide that

came with the BIOLOID PREMIUM kit. This section has some advanced instructional materials for humanoid gait generation using the Mirror Exchange function. If the reader had bought the PREMIUM kit before 2013, it would not contain this manual and it has to be bought separately at present (http://www.robotis-shop-en.com/?act=shop_en.goods_view&GS=1486&GC=GD080400).

Please visit the web site <http://thai.engr.uga.edu/RobotVids> for video clips of other legged robot projects from students.

6.4 Review Questions for Chap. 6

1. A MOTION file can be considered as DATA, while a TASK file can be considered as LOGIC. (T-F)
2. The Play Motion flag is set to 0 when a given Motion page is being executed. (T-F)
3. To control the accuracy of the Goal Position on an AX-12, we need to reduce its SLOPE value to the minimum value of 1. (T-F)
4. What does parameter PUNCH correspond to in functional terms for the motor component of a Dynamixel actuator?
5. What does parameter PRESENT LOAD correspond to in functional terms for the motor component of a Dynamixel actuator?
6. What does parameter SLOPE correspond to in functional terms for a Dynamixel actuator?
7. What are the ranges of possible values for the “Torque Limit” and “Present Load”?
8. What is the physical unit that parameter SLOPE can be mapped to?
9. What is the physical unit that parameter MARGIN can be mapped to?
10. How does one program the AX-12 to go to its “position control” mode from inside a TASK program?
11. Please match typical servo parameters to their correct addresses:

• Present Load	Address 30
• Present Position	Address 34
• Goal Position	Address 36
• Torque Limit (Goal Torque)	Address 40

12. What is the allowable range of angular positions in degrees for the AX-12 when it is set into Position Control mode?
13. What is the allowable range of angular positions in degrees for the XL-320 when it is set into Position Control mode?
14. When using the Motion Editor V.1 tool, what is the maximum number of steps/poses that the user can create per Motion Page, for each type of CM-5XX controller?
15. How many Motion Pages are allowed for each type of CM-5XX controller?
16. Describe in your own terms what is a POSE?
17. Describe in your own terms what is a STEP?

18. Describe in your own terms what is a MOTION PAGE?
19. How can a robot flow from one MOTION PAGE to another MOTION PAGE?
20. What is the EXIT PAGE used for?
21. To control the acceleration/deceleration patterns for a robot to go from Pose to Pose, we need to adjust the “Joint Softness” parameter of the servos involved in the robot’s moves. (T-F)
22. What is the effect of a small CONTROL INERTIAL FORCE setting?
23. What is the effect of a large JOINT SOFTNESS setting?
24. When using the Motion Editor V.1, for each specific step within a Motion Page, there is a value that ones can set for the “PAUSE” field, does that “PAUSE” happen “before” or “after” the execution of that specific step?
25. Describe the procedure to calibrate a multi-link robot using a specially designed pose for this task.
26. What is the common linkage system used to convert a rotational motion into a back-and-forth motion?

6.5 Review Exercises for Chap. 6

1. The enclosed program “AX-12ChangingModes.tsk” demonstrates how to change an AX-12 to its Continuous Turn mode (set 0 to Address 8) and then to its Position Control mode (set 1023 to Address 8).
2. The enclosed program “AX-12MonitorPositionSpeed.tsk” is a basic data acquisition program that prints data onto the TASK tool’s Output Window that the user can cut and paste into a data plotting application such as MS Excel.
3. The enclosed programs “PRM_GerwalkDemo.mtn” and “PRM_GerwalkDemo.tsk” work together to make a PREMIUM GERWALK (see Fig. 6.13) walk and avoid obstacles autonomously. Please modify the TASK code so that the user can control its movement via the RC-100, while keeping intact its obstacle avoiding functions.
4. Create a new Motion file (*.MTN) that would allow a GERWALK to kick a tennis ball. See enclosed video clip “Video 6.5” for several student solutions to a soccer penalty-kick simulation.
5. Create a new Motion file (*.MTN) that would allow a GERWALK or a BIPED to go up and down stairs steps:
 - Practice in generating appropriate motion pages combining CG shifting forward and up, backward and down as well as sideways recovery.
 - Alex Fouraker’s 2008 GERWALK solutions up and down six stair steps (see enclosed video files “Video 6.6” and “Video 6.7”—this one used the AX-S1 to check for a step existence first before stepping up).
 - Matthew Paulishen’s 2011 solution for a BiPed going up six stair steps (see enclosed video file “Video 6.8”).
 - Also enclosed is a sample MTN file “BipedUpStairs.mtn” that the user could start from.

6. Compare the servo configurations between the BIOLOID HUMANOID type A and the ROBOTIS-MINI.
7. Contrast similarities and differences about how the three actuators were used between the two designs: BIOLOID HEXAPOD and TWITCH?

References

- Abdel-Malek KA, Arora JS (2013) Human motion simulation. Academic, Waltham
- Chevallereau C et al (2009) Bipedal robots. Wiley, Hoboken
- Kajita S, Espiau B (2008) Legged robots. In: Siciliano B, Khatib O (eds) Springer handbook of robotics. Springer, Heidelberg, pp 361–389
- Kajita S et al (2014) Introduction to humanoid robotics. Springer, Heidelberg
- Kemp CC et al (2008) Humanoids. In: Siciliano B, Khatib O (eds) Springer handbook of robotics. Springer, Heidelberg, pp 1307–1333

Chapter 7

Advanced Position Control

Chapter 6 showed the “forward-control” features of the ROBOTIS RoboPlus V.1 software suite such as “Goal Position”, “Torque Limit” and “Motion Page”. In this Chap. 7, “Torque Limit” would be revisited but in a more “dynamic” manner allowing a limited feedback control capability. Additionally new control parameters such as “Present Load” and “Joint Offset” would be described and their usage demonstrated using an AX-12 based gripper and a special Function named “CALLBACK” from inside the TASK tool.

This chapter’s main topics are listed below:

- Concepts of “Torque Limit”, “Present Load”, and “Joint Offset”.
- Interactions between various actuator parameters: Goal Position, Slopes, Margins, Punch, Present Position, Present Load, Torque Limit and Join Offset.
- Use of “CALLBACK” function inside the Task tool.
- Illustrations of above concepts to a “load-sensing” gripper.

For more advanced robotic control topics, the reader is referred to Jazar (2010), Niku (2011) and Burdet et al. (2013).

7.1 “Torque” Effects

The goal of this section was to illustrate the interactions between the many parameters that influenced the process of moving a typical actuator (AX-12) from one Goal Position to another using a GERWALK as the test robot.

Electronic supplementary material: The online version of this chapter (doi:[10.1007/978-3-319-20418-5_7](https://doi.org/10.1007/978-3-319-20418-5_7)) contains supplementary material, which is available to authorized users.

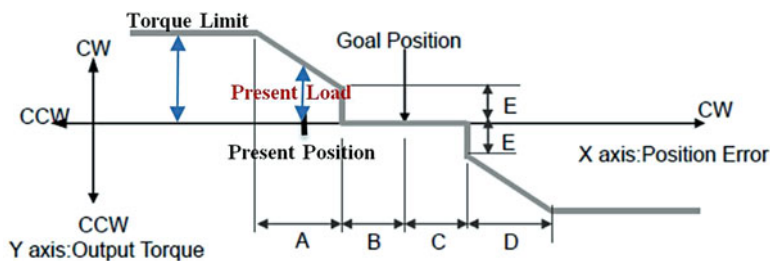


Fig. 7.1 Concept of present load

7.1.1 Torque Limit, Present Position and Present Load

Figure 7.1 was essential to the understanding of the interplay between the three parameters “Torque Limit” (Address 34, also labeled as Goal Torque in TASK), “Present Position” (Address 36) and “Present Load” (Address 40):

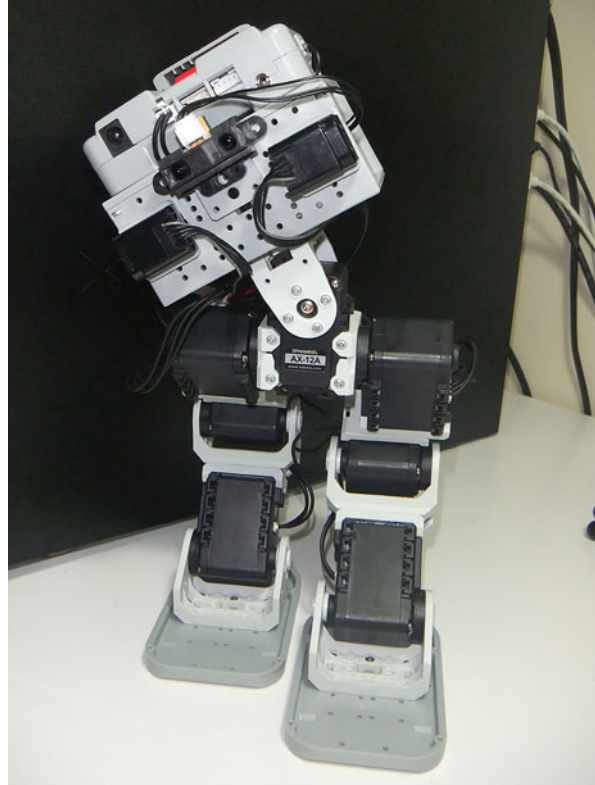
1. As previously described in Sect. 6.1, “Torque Limit” [0–1023] denoted the maximum load (electrical current) allowed on the actuator. When it was set high (512–1023), the user would need quite substantial physical effort to manually move the actuator from its current Goal Position. When it was set low (1–10), the user would be able to manually shift the actuator away from its Goal Position, and the actuator would not be able to get back to its Goal Position at all (this feature would be used later in Chap. 8 to illustrate the concept of “Bilateral Control” between two GERWALK robots).
2. Despite its given name of “Present Load”, this parameter should not be understood as the true external load on an actuator. More accurately, it would correspond to the “present electrical current” that was supplied to the motor by the actuator’s circuitry as long as the controller realized that the actuator’s Present Position was not where it was supposed to be (i.e. Goal Position \pm Margins B and C).

The video file “Video 7.1” and the TASK file “Gerwalk_TorqueEffects.tsk” provided a more thorough demonstration for these characteristics by monitoring the Present Position and Present Load during a manually forced resetting of the Goal Position(s) of the GERWALK actuators.

7.1.2 Adjusting Torque Limit Dynamically

In this subsection, the TASK files “Gerwalk_LoadAdjust.tsk” and “Gerwalk_LoadAdjustFast.tsk” were used to demonstrate the external behavior of actuators 4 and 5 of a GERWALK robot (Fig. 7.2) when they were manually forced out of their given Goal Position of 512. This would be the first step towards the

Fig. 7.2 Gerwalk robot used for adjusting torque limit dynamically



understanding of a feedback control approach for actuators such as the AX-12 (see Sect. 7.3 for a load-sensing application).

“Gerwalk_LoadAdjust.tsk” first initialized all seven AX-12s to a GOAL POSITION of 512 with a TORQUE LIMIT of 1023 during the first 2 s of its run, then it set the TORQUE LIMIT to 1 for Actuators 4 and 5 (to enable the user to manually move these two actuators out of Position 512 at his or her discretion). Next, it got into an endless loop where it printed out the following parameters: actuator’s ID, Torque Limit (Address 34), Present Position (Address 36) and Present Load (Address 40). Also within this endless loop, if the controller found that the Present Position of either actuator (4 or 5) was outside of a High-Low boundary around the initial position, it would increase the corresponding Torque Limit by a constant value saved in a parameter called TorqueStep. The net physical result was that one would see the actuators 4 and 5 go back to Position 512 at a constant rate, once the user “disturbed” their initial positions (see “Video 7.2”).

“Gerwalk_LoadAdjustFast.tsk” was designed to do the same job as “Gerwalk_LoadAdjust.tsk”, except that its TorqueStep value was proportional to the Gap between the actuator’s Present Position and either High or Low boundary values set around the Goal Position (see “Video 7.3”).

7.2 “Joint Offset” Effects

In Sect. 6.3.2, the video file “Video 6.2” showed the reader how to adjust the “Offset” values of individual actuators so as to correspond to a given Calibration Pose from inside the Motion Editor tool. In this Sect. 7.2, this JOINT OFFSET parameter would be re-visited as an advanced programming technique from inside the TASK tool and it would be applied to a single-actuator gripper (Fig. 7.3).

Fundamentally, the JOINT OFFSET parameter of a given actuator was designed to work only in conjunction with a MOTION PAGE (involving that particular actuator) being “played”. Please note that the GOAL POSITION parameter (Address 30) would not be affected in any way by the JOINT OFFSET parameter.

The possible range of values for JOINT OFFSET would be from -255 to $+255$ and essentially it was “added” to a given GOAL POSITION **as specified in a STEP** of a particular MOTION PAGE (and **not** by a regular LOAD command into Address 30, please make a note of this). For example, if JOINT OFFSET was set as -10 for a particular actuator, and a given STEP or MOTION PAGE commanded this actuator to go to Position 200, the actuator would actually go to a physical Actuator Position equal to $(200 - 10 = 190)$.

The “BasicGripper.mtn” and “AX-12JointOffset.tsk” files, along the video file “Video 7.4” demonstrated the above interactions for a single-actuator gripper. The reader could see that the invoked PAGE 3 (inside the endless loop) commanded the actuator to go to Position 512, but the actuator was actually decrementing its Goal Position by -10 every time that the program went through this loop (i.e. it kept on “opening up” the gripper).

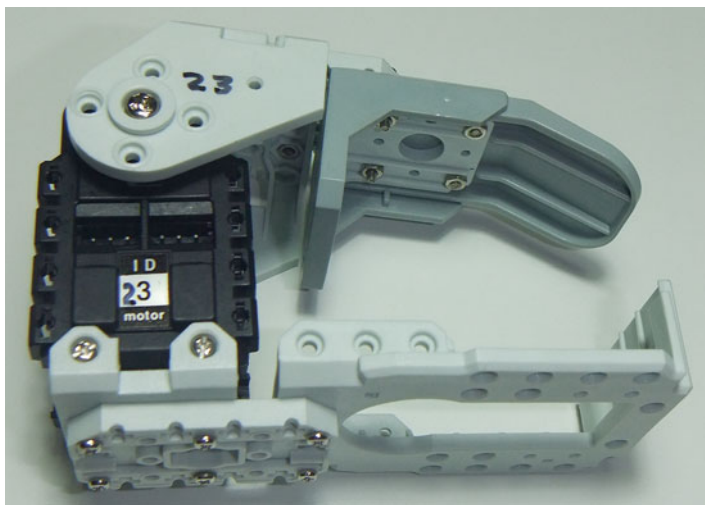


Fig. 7.3 A single-actuator Gripper

Finally, the reader should note that there was a special value of 1024 that could be set to the JOINT OFFSET parameter of a particular actuator, when the programmer wanted to **exclude** this particular actuator from the effects of a MOTION PAGE being executed. The ROBOTIS e-Manual web site has an application of this feature to the arms and grippers of a humanoid (Type B) robot (http://support.robotis.com/en/product/bioloid/premiumkit/tutorial/bioloid_prem_tutorial_gripper.htm). Please note that Dynamixel IDs 9 and 10 are assigned to the grippers in this particular example, and refer to the sample program files “PRM_HumanoidTypeB.mtn” and “PRM_GripperExam.tsk”.

7.3 A Load Sensing Gripper

In this section, the goal was to program a simple gripper that could adjust to different sizes of the object to be held by its fingers.

The software solution involved using the same “BasicGripper.mtn” file but a new TSK file called “AX-12Gripper.tsk” whereas the basic algorithm was described below:

1. Initialization to put the gripper at the neutral position by playing Motion Page 3, i.e. its Goal Position was set to 512.
2. Next the program entered an endless loop whereas:
 - (a) If the user pushed on Button U of the Virtual RC-100, the controller played MOTION PAGE 2 (i.e. Actuator Position set to 430—open gripper position).
 - (b) Else if the user pushed on Button D of the Virtual RC-100, the controller played MOTION PAGE 1 (i.e. Actuator Position set to 600—closed gripper position).
 - (c) Else if the user pushed on Button 1 of the Virtual RC-100, the controller played MOTION PAGE 3 (i.e. Actuator Position set to 512—neutral position).
 - (d) Also within this loop the program printed out the actuator Present Position, Joint Offset and Present Load for monitoring purposes.
3. The “size-adjusting” algorithm was implemented by a CALLBACK function which was executed by the controller every 8 ms. The interested reader should check the ROBOTIS e-Manual for more details about CALLBACK at http://support.robotis.com/en/software/roboplus/roboplus_task/programming/command/roboplus_task_cmd_callback.htm. This algorithm essentially read out the Present Load (Address 40) and compared this value:
 - (a) To “128”. If it was less or equal to 128, parameter Adjustment was set to 0 (i.e. the gripper had not encountered the object yet, or it was only starting to squeeze on the object).

- (b) To parameter `MaxSqueezeTorque` ($=750$), if it was greater than `MaxSqueezeTorque`, meaning that it was squeezing “hard” on the object then parameter `Adjustment` was decremented by 1 (i.e. the gripper was made to “open-up”).
- (c) The final value for `Adjustment` (whether 0 or a negative value) was assigned to `JOINT OFFSET` to affect a controlled “opening and closing” of the gripper so as to maintain a `PRESENT LOAD` “around” the value set by `MaxSqueezeTorque`.

The video file “Video 7.5” illustrated how this application translated out in practice.

The reader should note that this section was a very basic attempt at “Force Control”, Villani and Schutter (2008) have a more thorough treatment of the issues involved.

7.4 Review Questions for Chap. 7

1. What is the cycle time for execution of the `CALLBACK` function?
2. How many `CALLBACK` functions can be used in a `TASK` program?
3. What are the two steps needed to exclude a Dynamixel actuator from the effects of `MOTION PAGES`?
4. If the `PRESENT POSITION` of an actuator is at 512 and considering that its `JOINT-OFFSET` is set at -100 , what is its final position when a `TASK` command is issued for it to go to `GOAL POSITION` 800?
5. What is the procedure to use to allow a manual adjustment of a typical Dynamixel actuator when it is powered?
6. The parameter “Present Load” can tell us about the current mechanical loading on a given actuator. (T-F)

7.5 Review Exercises for Chap. 7

1. Start from the program “Gerwalk_LoadAdjust.tsk”, modify it so that:
 - (a) It can also print a `TIMER` value along with the existing parameters. Use the High-Resolution Timer if you happen to work with a CM-530.
 - (b) Try different combinations of `SLOPE`, `MARGIN` and `PUNCH` values to see changes in performance of the robot.
 - (c) Capture the data streams and plot the data with respect to the Timer counts.
2. Start from the program “Gerwalk_LoadAdjust_Fast.tsk”, modify it so that:
 - (a) It can also print a `TIMER` value along with the existing parameters. Use the High-Resolution Timer if you happen to work with a CM-530.

- (b) Try different MARGIN values to see changes in performance of the robot. Also try different values for the Gap divisor (currently set at 2).
 - (c) Capture the data streams and plot the data with respect to the Timer counts.
3. Use the program “AX-12Gripper.tsk” with different value combinations of the parameters “GrippingPower” and “GrippingSoftness” and let the gripper “chomp” on your finger. Note the differences in feeling on your finger.

References

- Burdet E et al (2013) Human robotics. MIT, Cambridge
Jazar RN (2010) Theory of applied robotics. Springer, Heidelberg
Niku SB (2011) Introduction to robotics. Wiley, Hoboken
Villani L, Schutter D (2008) Force control. In: Siciliano B, Khatib O (eds) Springer handbook of robotics. Springer, Heidelberg, pp 161–185

Chapter 8

Wireless Communications Programming

For its popular robotics systems, ROBOTIS offered three types of wireless communication protocols, NIR, ZigBee and BlueTooth:

- NIR was the default mode for its remote controller RC-100/A/B (<http://www.robotis.us/rc-100b/>) using the NIR Receiver OIR-10. The user could add the ZIG-100 module to it to get ZigBee capabilities (1-to-1, many-to-one, or broadcast), or the BT-100/BT-210 to get BlueTooth capabilities (1-to-1).
- The CM-5 system could only use the ZIG-100 module (or NIR via its AX-S1 module), while the CM-510, CM-530 and OpenCM-904 systems could interface with all three types of communication hardware.
- For ZigBee interfacing from Windows PCs to ROBOTIS systems, the user needed to use the hardware combination of USB2DYNAMIXEL/ZIG2SERIAL/ZIG-100/ZIG-110A. For BlueTooth interfacing, the user could use any PC BlueTooth devices (V. 2.0 and above) and either BT-110A, BT-210 or BT-410 (available after Summer 2015 and see Sect. 3.2.3).

Although ROBOTIS' NIR, ZigBee and BlueTooth communications hardware were different, their software application approach was standardized and based on a 16-bit message wrapped up in a 6-byte communication packet (http://support.robotis.com/en/product/auxdevice/communication/rc100_manual.htm). Section 5.6 presented the basic communication concepts of transmitting and receiving data from the RC-100 (Virtual and Physical) on a 1-to-1 basis. Later in this chapter, we will take on more advanced concepts such as broadcast programming and message shaping.

This chapter's main topics are listed below:

- Hardware channel differences between ZIG-100 and ZIG-110A.
- Embedding signals to control multiple robots with a single RC-100 (NIR or ZigBee).

Electronic supplementary material: The online version of this chapter (doi:[10.1007/978-3-319-20418-5_8](https://doi.org/10.1007/978-3-319-20418-5_8)) contains supplementary material, which is available to authorized users.

- Message shaping concepts as applied to Leader/Follower grippers and robots (bilateral control), and to multiple-user situations.
- PC to robot communications via Manager tool and via C/C++ programming tools.
- Comparing ZigBee and BlueTooth performances.

8.1 ZigBee Broadcast Channel Differences

ROBOTIS designed their ZigBee modules to operate on four possible broadcast channels, 1–4. ZIG-100 is defaulted to Channel 1, but could be changed to the other channels by modifying the status of Pins 7 and 8 (http://support.robotis.com/en/product/auxdevice/communication/zigbee_manual.htm). On the other hand, ZIG-110A was defaulted to Channel 4 and it was unchangeable. A further complication arose if the user needed to have ZigBee communications on the PC side by using the combination USB2Dynamixel/ZIG2Serial/ZIG-100 through an USB port, as the ZIG2Serial could also be set into four possible broadcast channels by opening or shorting three resistors R5, R6 and R7 (http://support.robotis.com/en/product/auxdevice/communication/zig2serial_manual.htm and see Fig. 8.1).

When using the ZigBee modules, either ZIG-100 or ZIG-110A, in a 1-1 mode such as in this example:

- ZIG-100 on an RC-100 or on a combo as shown in Fig. 8.1 on a PC.
- ZIG-110A on a CM-510 or CM-530.

The user only had to make sure that the ZigBee IDs matched via the MANAGER tool or by programming them directly inside a TASK program (http://support.robotis.com/en/product/auxdevice/communication/zigbee_manual.htm).

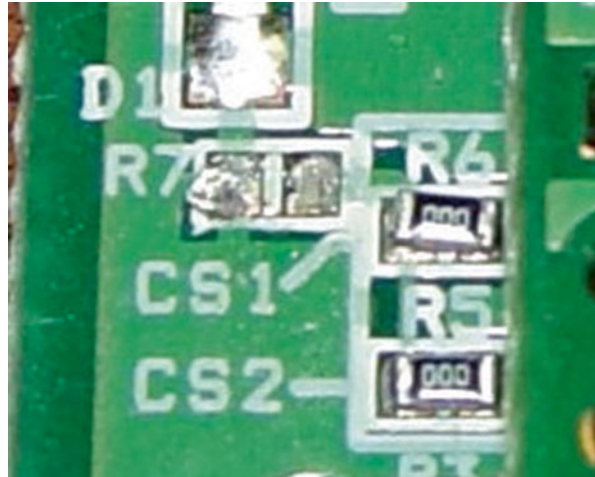
Also when the user needed to have broadcast ZigBee from the PC to other ZIG-100 modules, there would be no need for the user to do anything special at all as ZigBee modules involved were defaulted to Channel 1 already.

However when the user needed to have ZigBee broadcast capability from the PC to a set of ZIG-110A modules, the user needed to modify a ZIG2Serial module “permanently” in such a way the ZIG2Serial from then on would broadcast only on Channel 4 (see Fig. 8.2).



Fig. 8.1 USB2Dynamixel/ZIG2Serial/ZIG-100 combo

Fig. 8.2 ZIG2Serial module with R7 removed



8.2 Broadcast Use of RC-100 (NIR and ZigBee)

The goal of this section was to illustrate the extended use of a single RC-100 in the selective control of multiple robots. Please note that this project would work properly only when NIR or ZigBee communications hardware were used, as the BT-100/110A/210/410 hardware sets could only be used on a 1-to-1 basis (http://support.robotis.com/en/product/auxdevice/communication/bt100_110.htm).

A typical BIOLOID kit would come with an RC-100/A/B (http://www.robotis-shop-en.com/?act=shop_en.goods_view&GS=1487&GC=GD080302) and an NIR receiver OIR-10 (http://www.robotis-shop-en.com/?act=shop_en.goods_view&GS=1291&GC=GD080301). The user could set the RC-100/A/B in NIR mode to eight distinct 1-to-1 channels (http://support.robotis.com/en/product/auxdevice/communication/rc100_manual.htm). To complete the “communication loop”, the user would also need to set the matching channel inside the TASK program (for example, **RC-100 Channel = CH3** in case Channel 3 was used). However the special Channel 0 (**RC-100 Channel = CH0**) would make the robot “listen” to any channel, effectively allowing several robots to receive commands from a single RC-100 remote controller (and so, in a way, achieved broadcast control via NIR). The user should also take note that NIR communications was constrained physically to a “line-of-sight”, thus the group of robots would need to stay “close to one another”.

If ones used the ZigBee route, i.e., using ZIG-100/110A, the “line-of-sight” constraint would be removed (http://support.robotis.com/en/product/auxdevice/communication/zigbee_manual.htm). However, the setting of the “broadcast” mode (Remote ID=65535) on the ZIG-100/110A hardware would be more involved. The ZIG-100 module would need to be mounted on a ZIG2SERIAL+USB2DYNAMIXEL hardware combination (as modified in Sect. 8.1) and set to “broadcast ID” from inside MANAGER using its ZIG2SERIAL Management tool (see Fig. 8.3 and review the video file “Video 8.1” and the “TestZigBee.tsk” program).

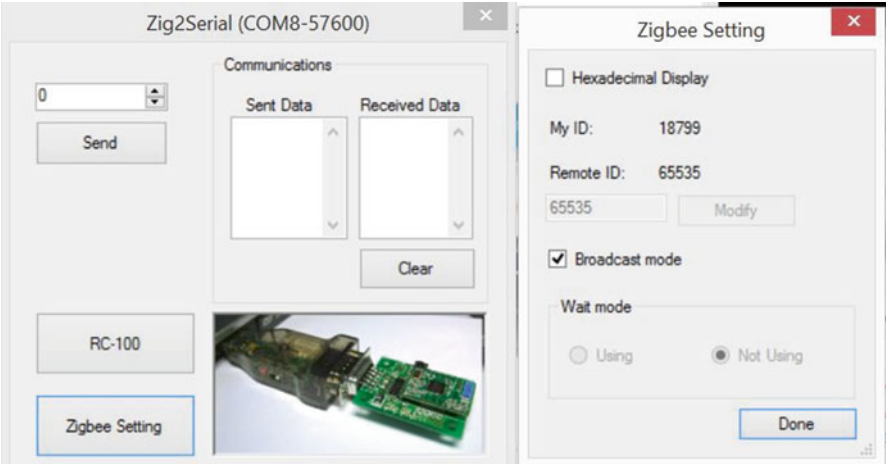


Fig. 8.3 Setting ZIG-100 to Broadcast mode (Remote ID=65535)

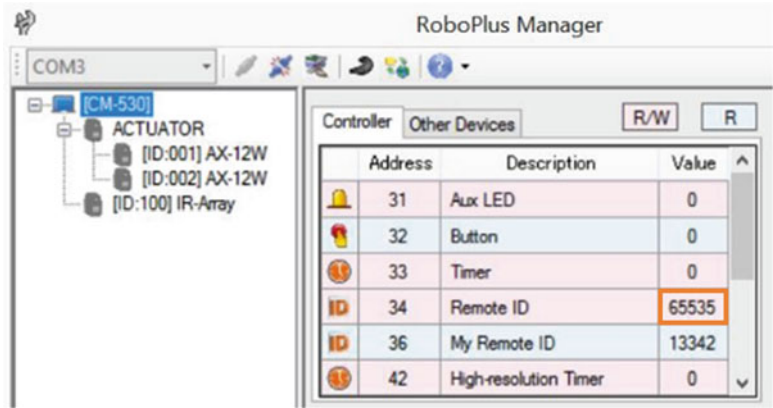


Fig. 8.4 Setting ZIG-110 to Broadcast mode (Remote ID=65535)

Furthermore a matching ZIG-110A module would need to be mounted on a typical CM-510/530 controller, and its “broadcast ID” (65535) would need to be set from the “Controller” panel inside the MANAGER tool (see Fig. 8.4). Currently, the MANAGER tool does not support such operation with the OpenCM-904-C controller.

The RC-100 had 10 buttons: U-D-L-R and numerical 1–6. In this application, the intent was to keep the U-D-L-R buttons for directional control of the movements of a typical robot, while the numerical buttons (1, 2, 3) were used to select a specific robot among three BIOLOID STEM robots (see Fig. 8.5): BugFighter (i.e., button 1), Droid (i.e., button 2) and Hexapod (i.e., button 3).

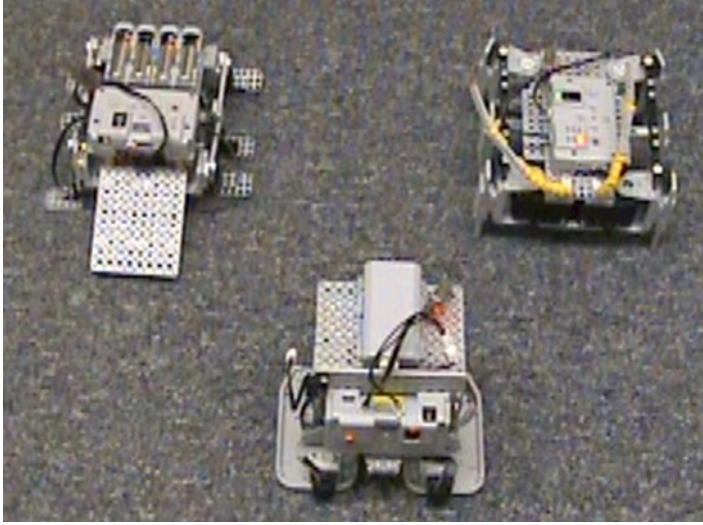


Fig. 8.5 Remote control of three BIOLOID STEM robots

To be exact, when the user used only the U-D-L-R buttons, all three robots would perform the same commanded maneuver, but if the user added one, two or all three (1, 2, 3) buttons, then only the selected robot(s) would perform the commanded maneuvers. Please review the four TASK programs (RC_1_BugFighter.tsk, RC_1_BugFighter_NIRReceiver.tsk, RC_2_Droid.tsk, RC_3_Hexapod.tsk) and also the video file “Video 8.2” for more details.

8.3 Message “Shaping” Concepts

When using the RC-100 as previously described, we were using only 10 bits out of 16 bits possible that were transmitted or received via ROBOTIS communication protocols (NIR, ZigBee or BlueTooth) within a **single** communication packet. In this section, several applications would be described, exploiting this capability:

- Mimicking grippers.
- Leader and Follower GERWALKs.
- Multiple users and multiple robots (ZigBee only).

8.3.1 *Mimicking Grippers*

In this sub-section, two applications of 1-to-1 ZigBee message-shaping techniques would be described using two single-actuator grippers, each controlled by its own CM-5XX controllers as shown in Fig. 8.6.

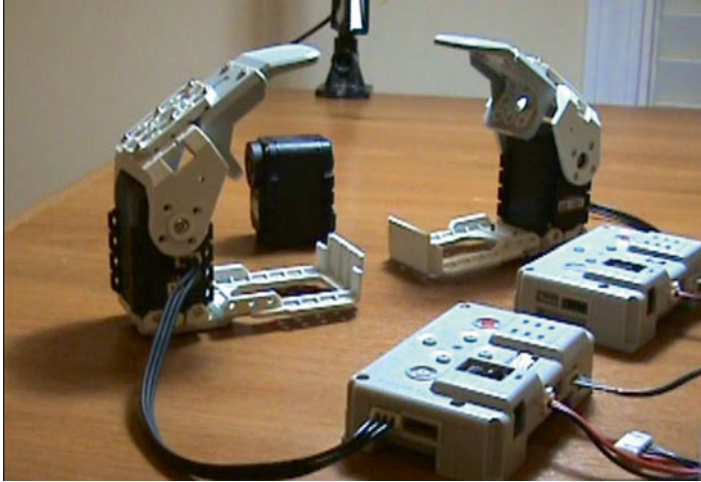


Fig. 8.6 Leader and follower grippers using 1-to-1 ZigBee

The first application was of an “Open Loop” system as the command structure went only one way—from Leader to Follower. Please review the TASK programs (“CM5XX-MimicGripperL_OpenLoop.tsk” and “CM5XX-MimicGripperF_OpenLoop.tsk”), MOTION files (“BasicGripper16.mtn” and “BasicGripper17.mtn”) and watch the video file “Video 8.3” for demonstration purposes, but the key concepts would be explained in the following paragraphs.

First, the only difference between the two MTN files mentioned earlier was that the Leader gripper had ID=16, while the Follower gripper had ID=17. The actual Motion Pages inside these MTN files had the same information:

- Page 1 (Close Grip) corresponded to Actuator Position 600 on the used AX-12s (ID=16 or ID=17).
- Page 2 (Open Grip) corresponded to Actuator Position 430 on either AX-12.
- And Page 3 (Neutral Pose) corresponded to Actuator Position 512 on the same AX-12s.

Next the feature called “Torque Limit” (or “Goal Torque”), first described in Sect. 7.1.2, was applied to both grippers but in a differential manner:

- The Leader gripper had its “Torque Limit” set to 1, so as to allow the user to manipulate this gripper as needed (Statement 26 in “CM5XX-MimicGripperL_OpenLoop.tsk”).
- The Follower gripper had its “Torque Limit” set to 1023 so as to allow it to do the real work (Statement 11 in “CM5XX-MimicGripperF_OpenLoop.tsk”).

The most important procedure was how to shape the 16-bit message so that a single communication packet could contain as many pieces of information that were needed to accomplish a given job. For this “Open Loop” application, the need was

for the Leader to send its Actuator ID (=16) and its Present Position (variable value) over to the Follower which would then mapped the Leader’s Actuator ID (=16) to match with its own corresponding ID (=17) and to set its Goal Position to the same value as the Leader’s Present Position as received in the 16-bit message:

- On the Leader side (“CM5XX-MimicGripperL_OpenLoop.tsk”), the goal was to configure the 16-bit message so that it contained the Actuator ID in the highest 6 bits and the Actuator’s Present Position (Address 36, and numerical range [0–1023]) in the lowest 10 bits. This meant that the ID value needed to be shifted 10 bits to the left of the 16-bit message which was accomplished by “multiplying” ID with the binary constant 0000 0100 0000 0000 (statement 31). Next the current Present Position value was added to the same 16-bit message (statement 33). This final combined message was then sent to the Follower with the “Remocon_TXD” function (statement 35). In the rest of this TSK file, minor procedures were written to cycle through all actuators used if more than one were used (statements 38–46), and to include some do-nothing code so as not to overwhelm the ZigBee channel (statements 49–53).
- On the Follower side (“CM5XX-MimicGripperF_OpenLoop.tsk”), the goal was to unpack the received 16-bit message so that it contained the Actuator ID in the highest 6 bits, by first bit-wise ANDing it with “1111 1100 0000 0000” (statement 23) and then by shifting right this temporary result by 10 bits with its division by “0000 0100 0000 0000” (statement 25). Statement 27 was used to adjust for the difference between the IDs used for the Leader and Follower grippers (the user may not have to perform this step if he or she chose the same ID for the actuators). Next the lowest 10 bits was filtered out into the parameter TargetPosition by ANDing the 16-bit message with “0000 0011 1111 1111” (statement 29). Lastly, a precautionary step was taken to check the TargetPosition’s value to see if it was less or equal to 1023 before actually assigning it to the Follower’s Goal Position (i.e., address 30 in statement 34).

The second application would be a “Closed Loop” system whereas information flowed both ways between Leader and Follower. Please review the TASK programs “CM5XX-MimicGripperL_ClosedLoop.tsk” and “CM5XX-MimicGripperF_ClosedLoop.tsk” and the video file “Video 8.4” for demonstration purposes, but the key concepts would be explained further in the following paragraphs.

In addition to the features/concepts already explained for the “Open-Loop” case, there were additional codes created to handle the bilateral flow of messages:

- On the Follower side (CM5XX-MimicGripperF_ClosedLoop.tsk), statements 40–76 were added to handle:
 1. Prepare the 16-bit message with the ServoID value in the highest 6 bits (statement 41).
 2. Save the Present Load value (address 40) into parameter LoadData and if LoadData was larger than LoadLimit (=512), i.e., there was some obstruction present, then the Follower needed to open up the gripper (statement 47) and sounded an audio alarm (statements 48–57). Next the Follower added its

LastPosition (address 36) into the 16-bit message and sent it away to the Leader (statements 58–61). Finally it went into a While Loop waiting for the user to push the “R” button on the CM-5XX controller (essentially to reset this alarm situation—statements 64–68). When “R” was eventually pushed, the Follower sounded the “All Clear” (statements 69–75) and got back to its Main Loop waiting for messages to come in from the Leader gripper.

3. If LoadData was less than LoadLimit, i.e., no obstruction encountered, the Follower just got back to its Main Loop and waited for messages to come in from the Leader gripper.
- On the Leader side (CM5XX-MimicGripperL_ClosedLoop.tsk), statements 59–102 were created to handle possible messages coming from the Follower (in case of an obstruction occurring to the Follower):
 1. After parsing the Follower’s 16-bit message into the ID and TargetPosition components (statements 61–69), the Leader “stiffened” up by setting its Torque Limit setting to 512 (statement 71) and by going to the “Open” position while sounding an audio alarm (statements 72–84). At this point, the user would no longer be able to manipulate the Leader gripper as before.
 2. Next it went into a While Loop waiting for the user to push the “R” button on the CM-5XX controller (essentially to reset this alarm situation—statements 86–90).
 3. When “R” was eventually pushed, the Follower went “soft” again by resetting its Torque Limit to 1 (statement 92) and sounded the “All Clear” (statements 95–101) and got back to its Main Loop generating 16-bit messages to send over to the Follower gripper, according to the user’s manipulations of the robot.

8.3.2 *Leader-Follower GERWALKS*

This sub-section extended the previous message-shaping and bilateral control concepts to a more complete robot such as the GERWALK. In this closed-loop application, both Leader and Follower robots were supposed to be ID’d identically (ID=1 to ID=7 in this particular case).

Similarly, please review the TASK programs (Gerwalk_L_ClosedLoop.tsk) and Gerwalk_F_ClosedLoop.tsk), GerwalkDemoMotion.mtn file and the video file “Video 8.5” for demonstration purposes, and the key concepts would be explained in the following paragraphs.

In this application, as we were dealing with seven actuators on the Leader robot, the overall plan was to send “ID” and “Present Position” information for only one actuator in each 16-bit message as before, and additionally used a scheme to cycle through the IDs from 1 to 7:

- On the Leader side (Gerwalk_L_ClosedLoop.tsk):
 1. Parameter ID started at 1 (statement 8) and all “Torque Limits” were set to 1 (statement 19) to allow user to manipulate the Leader robot at will.

2. The ID component was shifted to the highest 4 bits this time around (statements 23–25).
 3. Statements 27–34 were used to update Parameter ID and cycle it through from 1 to 7.
 4. The 50-count delay section (statements 35–40) was critical to the behavior of the Follower. For any shorter delay used, the Follower would lose its synchronization, as it could skip on the message received and consequently could also skip setting up properly one or more of its seven actuators.
 5. Statements 42–53 were used to decode potential messages from the Follower (when some or all its actuators detected an overload at a certain TargetPosition). In this situation, the Leader would “stiffen” up the corresponding actuator (defined by MotorID) by raising its Torque Limit value to 512 and by holding it at the TargetPosition sent over by the Follower (statements 48–49), but only for 0.128 s, and then by resetting it back to 1 (statements 50–52). The net “physical” result would be that the human user would feel this particular actuator “twitch” or “tremble” in his or her hand.
- On the Follower side (Gerwalk_F_ClosedLoop.tsk):
 1. Similarly as for the Follower gripper, statements 22–29 decoded the message from the Leader GERWALK and set the corresponding actuators to the TargetPosition values received.
 2. Statements 31–46 essentially cycled through the Follower’s seven actuators to see which one had its Present Load value (address 40) higher than the LoadLimit of 512. If such a situation arose, the Follower would transmit its own message to the Leader regarding this overloading actuator along with its ID and Current Position (address 36). Please note that this process was performed independently of the setting of the TargetPosition values as required in the messages sent by the Leader (previous paragraph 1.). In other words, messages were constantly “prepared” but not necessarily “sent” by the Follower to the Leader.

This sub-section was a very basic introduction to the bilateral control concept used often in tele-operation systems designed to work in hazardous environments or in medical robot systems. A few selected references are: Vertut (2012), Minh (2013) and Milne et al. (2013).

8.3.3 *Multiple Users and Multiple Robots (ZigBee Only)*

This sub-section extended the message-shaping technique developed by Mr. Matthew Paulishen (during his student days at my laboratory) for his Mobile Wireless Sensor Network project (see Thai and Paulishen (2011)).

Figure 8.7 illustrated the targeted situation where we would have several users (i.e., several RC-100s with ZigBee being used in broadcast mode) trying to control



Fig. 8.7 Multiple users controlling multiple twin GERWALKs

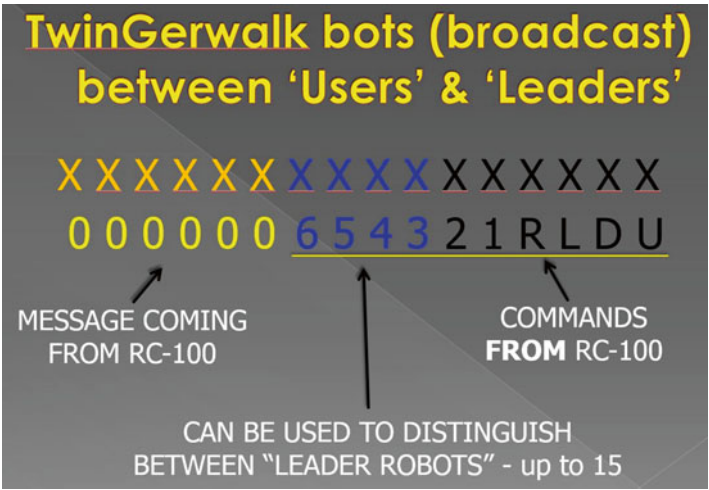


Fig. 8.8 Bit configuration for 16-bit messages broadcasted from RC-100s

their own robots (Twin GERWALK) which also had 2 independent controllers in their design (Leader and Follower, also set to broadcast mode). Please see enclosed video file “Video 8.6” for a solution by Duke TIP students.

Figure 8.8 described the detailed bit assignment when using the RC-100s to send messages out to Leader robots. Essentially, each user was allowed to use the U-D-L-R buttons for maneuvering and the numerical buttons 1 and 2 for special instructions (i.e., $2^2 - 1 = 3$ separate special instructions allowed). Each user used buttons

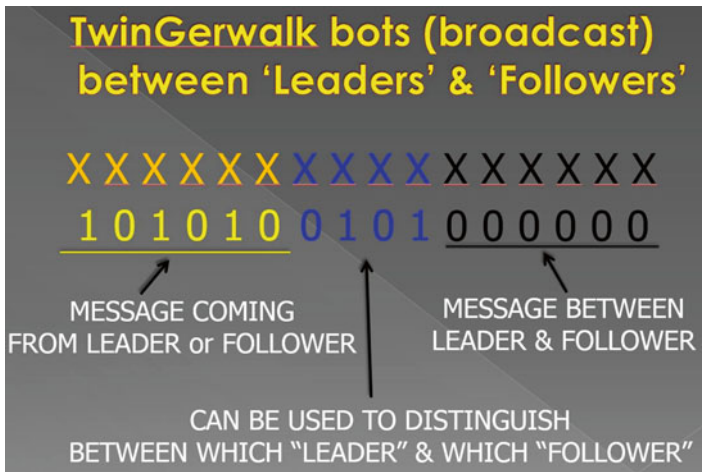


Fig. 8.9 Bit configuration for 16-bit messages broadcasted from Robots

3–6 to denote the specific Leader robots that would be assigned to particular users (with 4 bits, that would amount to 15 potential Leaders).

The reader should note that this was a broadcast environment so all the distinctions to be made had to be embedded in the 16-bit messages which were “constantly” broadcasted among the RC-100s and robots at play. Also the reader probably already noted the “honor” system that would have to be followed as any user could “jam” or “misdirect” the other user’s robot with the above scheme.

Figure 8.8 also showed that if a message came from the RC-100s, its highest 6 bits would be set to 0 and this feature would be used to distinguish the messages coming out the RC-100s from the ones coming out from either Leader or Follower robots as shown in Fig. 8.9.

Essentially, codes running on the robots would look for the pattern “000000”, “101010” or “010101” in the highest 6 bits of every message that they received:

- If this pattern was “000000”, then this message came from the RC-100s and would need to be processed further by the Leader robots **only**. Then the Leader robots would investigate the lowest 10 bits according to Fig. 8.8 to decode what actions their assigned users wanted them to do.
- When executing those user-assigned actions, the Leader robots would definitely need to send their own messages to their matching Followers to coordinate the required physical actions between Leader and (matching) Follower to perform a given maneuver. These messages would have the pattern “101010” embedded in the highest 6 bits. The lower 6 bits would contain the “true” message between matching Leader and Follower, while the mid-4 bits would be used to distinguish which Leader-Follower units (among the 15 possible) that this message pertained to.
- When the Follower robots needed to send their own messages to their matching Leaders, they would use the pattern “010101” in the highest 6 bits.

8.4 PC to Robots Communications via C/C++

If the reader wanted to do some communications programming between the PC and ROBOTIS' robots, but still stayed within the RoboPlus environment, then he or she could use the "Zig2Serial Management" sub-tool inside RoboPlus' Manager tool. As its name indicated, the "Zig2Serial Management" sub-tool was created to help the user with managing the Zig2Serial module when used with a ZIG-100. This sub-tool would allow the user to set the ZIG-100's communication mode (1 to 1, many to 1 and broadcast—as shown in the previous video file "Video 8.1") and also to send numerical data via the associated Windows COM port regardless of which wireless hardware was actually used—ZigBee or BlueTooth (as will be used in Chap. 11 for the ROBOTIS-MINI). As we were using the ROBOTIS' protocol with the standard 6-byte packet, we would also have to stay with the closed-firmware option on the robot side, i.e., operational with all the CM-5XX systems and OpenCM-904-C (closed-firmware mode), but not workable with OpenCM-904-A/B as they used OpenCM IDE.

If the reader wished to use C/C++ programming on the PC side for more flexibility, but still stayed with the closed-firmware option on the robot side to benefit from the TASK and MOTION features of the RoboPlus Suite, then he or she needed to have a closer look at the contents of the enclosed file "WirelessRobot.zip". This ZIP file contained all source codes and workspace files needed when using MicroSoft Visual Studio Express (2010, 2012 and 2013 editions and even Visual C V.6). This package was created by Mr. Matthew Paulishen with a simple command-line interface with the goal of introducing C/C++ constructs needed to properly integrate with the "ZigBee" and "Dynamixel" SDKs from ROBOTIS (http://support.robotis.com/en/software/zigbee_sdk.htm and <http://support.robotis.com/en/software/dynamixel/sdk.htm>). After unzipping this file, the folder "WirelessRobot\code\app\WirelessRobot DataLogger Binaries" contained the various executables for the DataLogger projects and the TASK files "WirelessCarbot.tsk" and "CM5xxDataLogger.tsk" for the reader to try out (*Note: Depending on what DLLs were installed on the reader's PC, not all executable programs would work properly, but the VC6 version should work for all Windows OS from XP to 8.1*).

The video file "Video 8.7" illustrated the use of the "DataLogger" executables with a CM-5 CarBot running the "WirelessCarbot.tsk" program via ZigBee hardware. First it demonstrated how the Zig2Serial Management sub-tool (of ROBOPLUS MANAGER) worked with the "WirelessCarbot.tsk" program. Next, it showed how the Visual C/C++ executable "DataLogger.exe" program worked with the same TASK program.

The much longer video file "Video 8.8" illustrated the various C/C++ constructs for understanding ZigBee Communications programming for Visual Studio Express 2012 (but it should be applicable to the 2013 edition also). This video also referred to the "CM5XXDataLogger.tsk" files.

8.5 ZigBee and Bluetooth Performances

ROBOTIS ZigBee hardware was of an older design so its maximum baud rate was 115,200 bps, while the latest BT-210 module could potentially go to 400,000 bps (but operated around 250 kbps) and besides Bluetooth devices are widely available in current PCs and mobile devices. Surprisingly, the newer BT-410 series would operate lower at 128 kbps to save energy on mobile devices. Thus if the reader plans to work with only one robotic system from ROBOTIS, and if he or she already has Bluetooth on their computing platforms, then Bluetooth is the obvious choice for accessibility and cost. However, Bluetooth usually takes rather a long time to connect as compared to ZigBee. Bluetooth also required two COM ports on the PC for each BT connection (one outgoing and one incoming) while ZigBee required only one COM port. This “feature” could become important if the user is interested in controlling teams of robots from a central PC. Hughes et al. (2013) shared some hands-on knowledge for Bluetooth applications to teams of LEGO NXT robots that the reader might find useful. Huang and Rudolph (2007) have essential information for BT programmers in Windows and Linux environments.

In Sect. 4.3, the author had shown that for situations with fast changing communication data, ROBOTIS ZigBee hardware/software outperformed ROBOTIS Bluetooth hardware/software (at least for now).

Lastly, from the point of view of educational flexibility, the option of changing to different ZigBee modes (1 to 1, many to 1 and broadcast) clearly favored ZigBee over Bluetooth. Thus overall, the author still preferred ZigBee over Bluetooth if the development environment can accommodate USB/ZigBee communications.

On mobile devices, all of us would have no choice but to use Bluetooth! The newer BT-410 is supposed to allow one master controlling several slave devices (due late Summer 2015), so another “seesaw” battle!

8.6 Review Questions for Chap. 8

1. List the three types of wireless communication protocols that are available with ROBOTIS systems.
2. How many NIR communication channels are available with the combination of RC-100 and OIR-10 modules?
3. Describe how to set up a chosen RC-100 Channel inside a TASK program.
4. The MANAGER tool can be used to set up ZigBee communication IDs (T-F)
5. The MANAGER tool can be used to set up Bluetooth communication IDs (T-F)
6. Describe how to set up a chosen RC-100 Channel on an RC-100 Remote Controller.
7. On ROBOTIS systems, NIR communications are strictly on a 1-to-1 basis (T-F)
8. On ROBOTIS systems, Bluetooth communications are strictly on a 1-to-1 basis (T-F)

9. Describe the current options for getting Bluetooth communications among ROBOTIS systems.
10. Describe the hardware hook-up needed to communicate between two CM-530s via Bluetooth communications.
11. Describe the hardware hook-up needed to communicate between a CM-530 and a PC via Bluetooth communications.
12. Describe the hardware hook-up needed to communicate between a CM-530 and an RC-100 via Bluetooth communications.
13. How many COM port(s) are open on the PC side for each Bluetooth module?
14. How many COM port(s) are open on the PC side for each ZigBee module?
15. Describe the hardware hook-up needed to communicate between two CM-5s via NIR communications.
16. Describe the hardware hook-up needed to communicate between two CM-5s via ZigBee communications.
17. Describe the hardware hook-up needed to communicate between a CM-5XX and an RC-100 via ZigBee communications.
18. Describe the hardware hook-up needed to communicate between a CM-5XX and a PC via ZigBee communications.
19. What are the three modes of communications for ROBOTIS ZigBee protocols?
20. Can the Zig2Serial sub-tool inside the MANAGER tool be used with Bluetooth communications between the PC and a robot?
21. Describe the standard packet configuration used by ROBOTIS for its wireless communications network.
22. Within a standard communication packet, how many bits does the actual “message” component contain?
23. Assuming that Parameter B = **0000 0000 0000 1011** , what is the result for Parameter A, after this TASK command was executed?

$$A = B \& \text{0001 0000 0000 1100}$$
24. Assuming that Parameter B = **0000 0000 0000 1011** , what is the result for Parameter A, after this TASK command was executed?

$$A = B * \text{0000 0100 0000 0000}$$
25. Assuming that Parameter B = **0110 0100 0000 0000** , what is the result for Parameter A, after this TASK command was executed?

$$A = B / \text{0000 0100 0000 0000}$$
26. Describe procedure(s) to achieve broadcast communications with ROBOTIS NIR hardware and software.
27. Describe procedure(s) to achieve broadcast communications with ROBOTIS ZigBee hardware and software.
28. Describe procedure(s) to achieve broadcast communications with ROBOTIS Bluetooth hardware and software.

29. Review the information found from the following web links and design an alternate approach to obtain ZigBee broadcast capability from the PC to a group of ZIG-110A devices:
- (a) http://support.robotis.com/en/product/auxdevice/communication/zigbee_manual.htm.
 - (b) http://support.robotis.com/en/product/auxdevice/communication/zig2serial_manual.htm.
 - (c) http://support.robotis.com/en/software/roboplus/roboplus_manager/testandconfigure/etc/roboplus_manager_zig2serial.htm.

8.7 Review Exercises for Chap. 8

1. Adapt the programs “Gerwalk_L_ClosedLoop.tsk” and “Gerwalk_F_ClosedLoop.tsk” to your own favorite multi-linked robots.
2. Add to the previous programs features of audio alarms using the ideas proposed in “CM5XX-MimicGripperL_ClosedLoop.tsk” and “CM5XX-MimicGripperF_ClosedLoop.tsk”.
3. Starting from the Visual C++ files inside “WirelessRobot.zip”, the goal of this assignment is twofold:
 - (a) To create a new TSK file “WirelessCarbot.tsk” that can accept “U-D-L-R” (for motion directions) and “1-2” (for Low-High speed settings) commands from the PC keyboard via ZigBee (as already shown in the example “CMDatLogger.TSK” and “main.CPP” files included in “WirelessRobot.zip”). Additionally and independent of the previous “direction” and “speed” commands, when receiving a special command from the PC, it should respond back to the PC with the designated NIR sensor reading(s) corresponding to the “Left-Center-Right” sensors of the AX-S1 module.
 - (b) To modify the “main.cpp” file so that it can do the following procedures:
 - It does not have to “echo” the various “U-D-L-R-1-2” commands onto the run-time PC display anymore, as the user can see those effects directly on the Carbot’s behaviors.
Initially, it should display an option menu telling the user various keyboard actions that are available to the user:
 - “W-A-S-D” keys for “U-L-D-R” directions.
 - “1-2” keys for carbot speed setting.
 - “O” for a request to do a 1-time scan (and PC-side display) of the user-hosen NIR sensor(s) on the Carbot.
 - “C” for a request to do a continuous scan (and PC-side display) of the **user-chosen** NIR sensor(s) on the Carbot.

- When either “O” or “C” options are in effect, the PC-side program should also ask the user for “how many” and “which” sensors for the Carbot to scan and transmit the data back to the PC.
 - “S” should stop the “continuous scan” mode.
4. Mobile Wireless Sensor Network Project. Using the Thai and Paulishen (2011) IEEE paper, design a wireless mobile sensor network with one PC acting as the control station, and three Carbots acting as mobile scouts that can send back to the control station data from its three NIR sensors located in the AX-S1 module. The 2011 IEEE paper describes in fairly good details a possible approach to be used that you can adapt or come up with your own scheme as you wish:
- Your system should be able to issue commands for the three Carbots to disperse and spread themselves out as far as possible from the base station, but without losing ZigBee communications with each other (PC and Carbots). Essentially, the Carbots will string themselves out so as to form a relay system whereas the furthest-out Carbot will send its data to the middle Carbot which would then relay those messages to the one nearest to the base station which would next relay those messages to the base station as the last step. A similar relay scheme should be applicable for the middle and nearest Carbots with less data hops to perform.
 - The base station would then display on the PC display the information that it got from each specific Carbot.
 - Not implemented in the 2011 IEEE paper was the capability to steer a particular Carbot from the PC (a laptop would be best)—as the human user would be able to see where the Carbots were going down a hallway for example (see enclosed video file “Video 8.9” to see a completed project).

References

- Huang AS, Rudolph L (2007) Bluetooth essentials for programmers. Cambridge University Press, Cambridge
- Hughes C, Hughes T, Watkins T, Kramer B (2013) Build your own team of robots with LEGO MINDSTORMS NXT and Bluetooth. McGraw-Hill, New York
- Milne B et al (2013) Design and development of teleoperation for forest machines: an overview. In: Habib MK, Davim JP (eds) Engineering creative design in robotics and mechatronics. IGI Global, Hershey, pp 186–207
- Minh VT (2013) Development and simulation of an adaptive control system for the teleoperation of medical robots. In: Habib MK, Davim JP (eds) Engineering creative design in robotics and mechatronics. IGI Global, Hershey, pp 173–185
- Thai CN, Paulishen M (2011) Using ROBOTIS BIOLOID systems for educational robotics. http://thai.engr.uga.edu/PDF/SE_IEEE_2011.pdf. Accessed 29 Dec 2014
- Vertut J (2012) Teleoperation and robotics: applications and technology. Springer, Heidelberg

Chapter 9

Advanced Sensors

In Chap. 3, advanced sensors such as the AX-S20, GS-12, FPS and HaViMo2 were briefly described. In this chapter, more application details would be provided whereas the Callback function was a critically needed tool.

This chapter's main topics are listed below:

- Static balance of humanoid robot using the AX-S20 (2-Leg and 1-Leg versions).
- Dynamic balance of humanoid robot using the GS-12 for Walking Enhancement and Fall Detection.
- FPS application to a humanoid robot's feet (1 Leg balance).
- HaViMo2 application to CarBot.

9.1 Humanoid Static Balance with AX-S20

In this section, a Humanoid A robot from the BIOLOID PREMIUM kit was used as the test platform (see Fig. 9.1). It actually had the AX-S20 mounted in its head and the GS-12 mounted inside the robot, just above the hip's actuators.

As the AX-S20 had a magnetometer, it had to be installed as far away as possible from the actuators which were generating strong and fluctuating magnetic fields which would interfere with the workings of this sensor. Thus it was mounted in its head (fortunately there was enough room in it). On the other hand, as the GS-12 (designed by ROBOTIS Vice-President In-Yong Ha) was based on MEMS sensors, there was no need to worry about magnetic interferences from the actuators, so it was mounted as much as possible near the center of gravity of the robot (i.e., on top of the hip's actuators).

Electronic supplementary material: The online version of this chapter (doi:[10.1007/978-3-319-20418-5_9](https://doi.org/10.1007/978-3-319-20418-5_9)) contains supplementary material, which is available to authorized users.

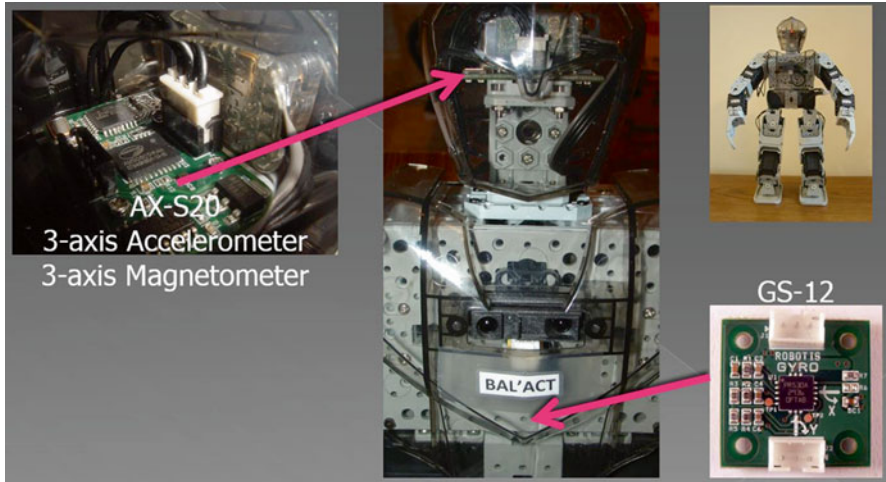


Fig. 9.1 Premium Humanoid A with AX-S20 and GS-12

9.1.1 2-Leg Static Balance with AX-S20

Figure 9.2 showed how the AX-S20 would appear inside the MANAGER tool as a Dynamixel-compliant sensor. Please note the various parameters (i.e., addresses) accessible to the user from inside a TASK program (using “Custom” commands with a Word size):

- Azimuth angle with respect to Z-axis, in degrees [0–359] (address 26).
- Pitch (with respect to Y-axis) and Roll (with respect to X-axis) angles in degrees, [–89 to 89], at addresses 28 and 30 respectively. Please see Fig. 9.3 for the coordinates system used by the AX-S20.
- “Real-time” X-Y-Z acceleration values could also be read via addresses 38, 40 and 46 respectively, [–2048 to 2048], i.e., at 0.01225 m/s^2 per count. The AX-S20 has about a 20 Hz refresh rate (i.e., rather slow, as it was designed around 2009).

The first AX-S20 application derived from a ROBOTIS demo program (c. 2009) which described how to read its Pitch and Roll angles and use those real-time data to adjust the Joint Offsets of selected actuators of both legs so that the Humanoid robot could maintain its “static” balance even though the platform, where it was standing on, was changing its inclination angles with respect to the Pitch and Yaw axes (see video file Video 9.1 and Fig. 9.4).

This application used the following files: “BalAct_2Legs_AXS20_Balance.tsk” and “BalAct_2Legs.mtn”. The basic steps in the balance control algorithm used were as follows:

1. Get the robot into the initial pose as smoothly as possible by “playing” Motion Page 1 and then Motion Page 129 (statements 6–9 in the “BalAct_2Legs_AXS20_Balance.tsk” file).

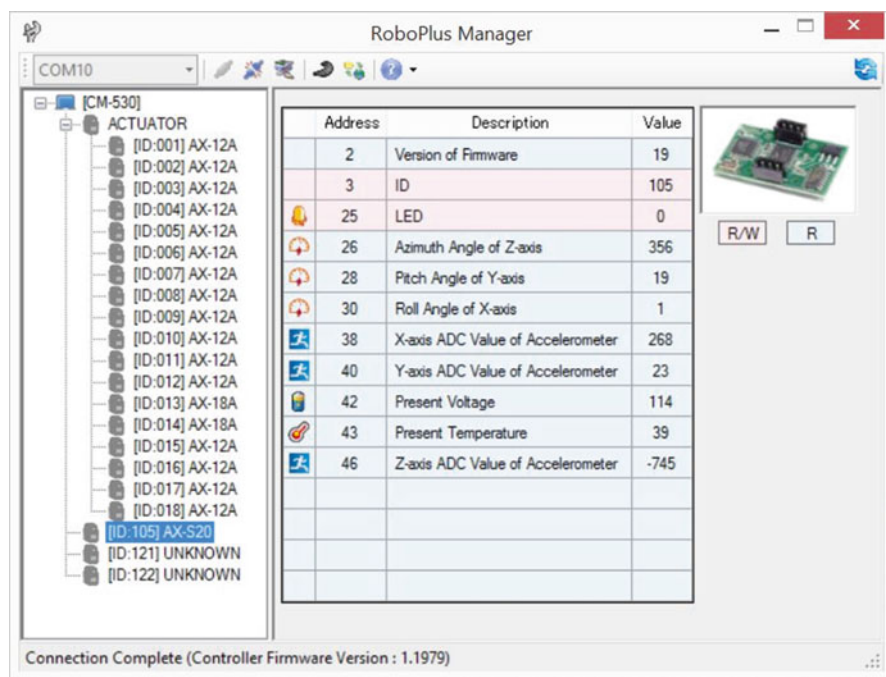


Fig. 9.2 AX-S20 panel inside MANAGER tool

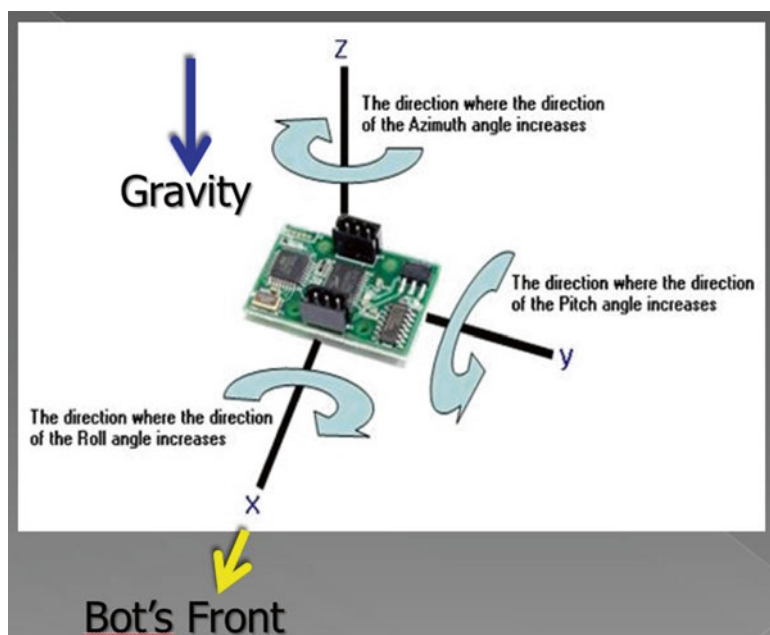


Fig. 9.3 Coordinates system used by the AX-S20

Fig. 9.4 Humanoid Type
A 2-leg balancing using
AX-S20



2. Collect 10 consecutive values of the X and Y components of gravity (addresses 38 and 40) and save their average values respectively into the parameters “FBBalCenter” and “RLBalCenter” (statements 10–20).
3. Next play Motion Page 128 (balance position at statement 27) which had the same Goal Position values for all the actuators as Motion Page 129, except that “128” would keep on calling itself indefinitely, essentially enabling the Joint Offset functions to work in maintaining the robot balance position, when the user started to modify the inclination angles of the supporting platform, as shown in the video file “Video 9.1”.
4. Effectively, “FBBalCenter” and “RLBalCenter” became the set points for this control algorithm which would seek to minimize future deviations of these values by triggering appropriate Joint Offset Values of the following actuators with ID numbers:
 - a. (11, 13, 15) for the right leg and (12, 14, 16) for the left leg. Please note that these actuators influenced most directly the forward-backward motions (see Fig. 9.5).
 - b. (17) and (18) which respectively influenced the right-left motions of the “ankles”.
 - c. In other words, this robot would seek to balance itself by adjusting its “ankles” and “knees” and by “crouching” up or down differentially between its right and left legs. Of course other solutions would also be possible by bringing actuators 9 and 10 in the mix for side-shifting of the hip.

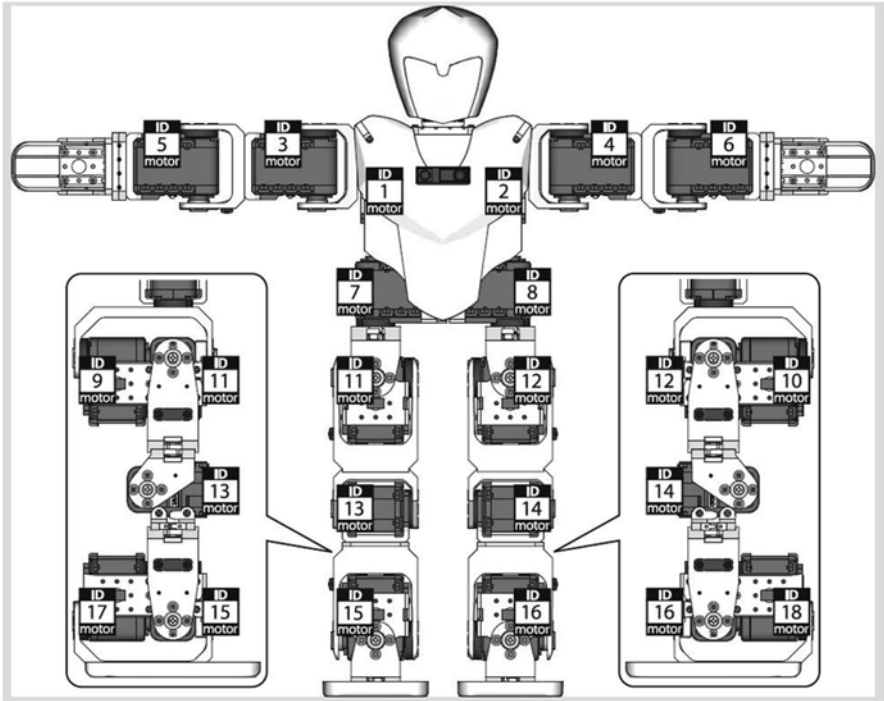


Fig. 9.5 IDs of actuators used for the BIOLOID Humanoid Type A robot

5. This control algorithm was actually carried out “in parallel” of the main program by a CALLBACK function (statements 47–117) (http://support.robotis.com/en/software/roboplus/roboplus_task/programming/command/roboplus_task_cmd_callback.htm). ROBOTIS designed CALLBACK to be activated every 8 ms which happened to be the hardware refresh cycle time for all Dynamixel. Because of this rather short time period, the programmer could not use logical constructs such as loops and was limited to a maximum of two hardware calls, and furthermore the size of the CALLBACK function could not exceed 512 bytes.

The main sections of the CALLBACK function were as follows:

- Re-init “FBBalCenter” and “RLBalCenter” (statements 54–55).
- Read in new data for “FBBalData” and “RLBalData” (statements 59–60).
- Compute the current errors, “FBBalError” and “RLBalError” (statements 63–64) and ignore small errors by dividing them with 16, i.e., low byte discarded, to get the “Scaled” error parameters (statements 67–68).
- Next, these “Scaled Balance Error” parameters got “summed up” (i.e., integrated over time) with an upper limit (=2048) and a lower limit (= -2048) for both the FB and RL “Scaled Balance Error” parameters (statements 70–88).

- e. Statements 91–104 represented a complex “Gain Application” algorithm that sought to convert the “Balance Error Sums” parameters (i.e., angle values in degrees) into “appropriate” AX-12 Joint Offset values [–255 to 255] that depended on which part of the leg did that particular servo belong to. For example, adjustments to the “side-to-side ankle” servos (IDs 17 and 18) depended only on the “RL” errors and were numerically small as they represented the pivot point of an “inverted pendulum” (statements 92, 98, 102, 115 and 116). While adjustments to the “hip”, “knee” and “forward-backward ankle” servos (IDs from 11 to 16) depended on both “FB” and “RL” error terms (statements 107–113), and they were numerically larger as they were further away from the pivot point of the “inverted pendulum”. The reader could also note the “mirror image” effect used in the mathematical expressions used for the “right” leg (statements 107–109 for servos 11, 13 and 15) vs. the “left” leg (statements 111–113 for servos 12, 14 and 16). For more advanced analysis of bipedal walking motions, the reader would need to consult such works as Chevallereau et al. (2009) or Kajita et al. (2014).

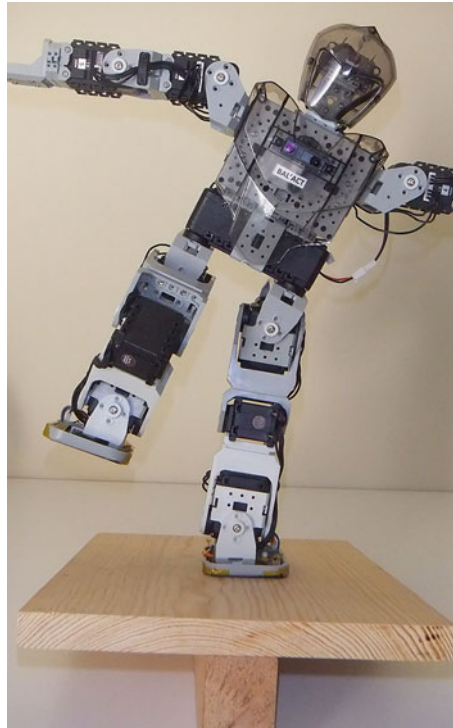
9.1.2 1-Leg Static Balance

This work was done by Mr. Matthew Paulishen during his UGA student years. It used the two files “BalAct_1Leg_AXS20.mtn” and “BalAct_1Leg_AXS20_Balance.tsk” (see Fig. 9.6 and video file “Video 9.2”).

This application did all the data acquisition and joint-offset computing inside the main TSK program and no CALLBACK function was used:

1. After zeroing out all previous joint-offsets (statement 13), this TASK program waited for the user to push the “Up” button for the first time to set the robot into Motion Page 13 which made the robot stand on its left leg while curling up its right leg. The user could now set the 1-legged robot onto the test platform (statements 18–22).
2. When ready to begin the actual balancing act, the user would push the “Up” button for the second time (statements 25–27) to trigger the function “ReCenterAXS20” (statements 128–154) which would acquire ten consecutive values of the FB and RL acceleration parameters (addresses 38 and 40 respectively) and saved the average values into “FBAccelCenter” and “LRAccelCenter” parameters (statements 148–149).
3. The robot was then ready to play Motion Page 12 (which called itself indefinitely) and got into an infinite loop to balance itself (statement 34–96) as the user changed the angles of the platform triggering changes in the values of the FB and RL acceleration parameters.
4. Similarly to the approach used in Sect. 9.1.1, the FB and RL acceleration error terms were then converted into appropriate joint-offsets for the legs’ servos. Please note that while the “left knee” (ID= 14) was locked in place, the “left hip”

Fig. 9.6 Humanoid Type
A 1-leg balancing using
AX-S20



(IDs= 10, 12) and “left ankle” (IDs= 16, 18) were “key players” in this balancing act, while the “right leg” lent its support via servos 9 and 11 for larger FB and RL error values, i.e., for larger angles of the standing platform (statements 77–90).

As previously mentioned in Chap. 3, the AX-S20 is no longer available from ROBOTIS, and the author is not aware if ROBOTIS is working on a similar sensor for the CM-5XX series. For the OpenCM-9.04-B controller, a more recent inertia-measuring device such as the SEN-11486 (MPU-9150) (<https://www.sparkfun.com/products/11486>) was incorporated into the OpenCM IDE V.1.0.2, but so far the author is not aware of an adaptation of this IMU device to the OpenCM-9.04-C or the CM-5XX series.

9.2 Humanoid Dynamic Balance with GS-12

A “2-legs balance” application, similar to the one made for the AX-S20 (see Sect. 9.1.1), could be created for the GS-12 also. Please review the enclosed files “BalAct_2Legs.mtn” and “BalAct_2Legs_Gyro_Balance.tsk”. The enclosed video clip “Video 9.3” explained the algorithm used and showed the performance obtained on a Humanoid Type A and a CM-530 controller.

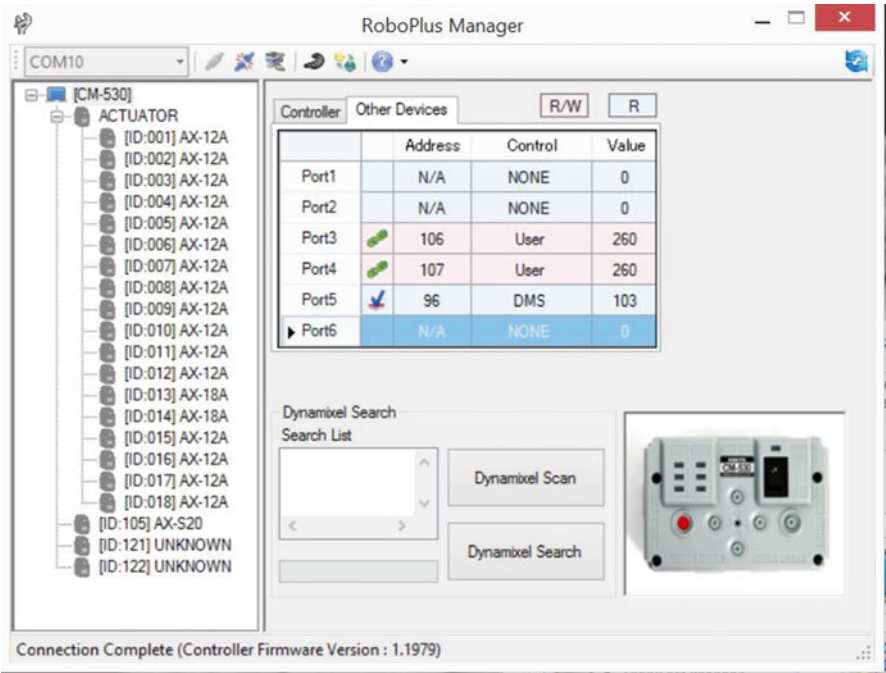


Fig. 9.7 GS-12 panel inside MANAGER tool

A more extensive application of the GS-12 sensor to the operations of a Humanoid A robot was also illustrated in the example TSK and MTN files provided by ROBOTIS (see files “BIO_PR_M_Humanoid_A.tsk” and “BIO_PR_M_Humanoid_A.mtn”). Figure 9.7 showed Ports 3 and 4 being used for the GS-12 in our demonstration robot Bal’ Act.

The GS-12 measured angular rates with respect to the (X, Y) axes (see Fig. 9.8) and the range of possible values for its outputs were from “45” (i.e., $-300^{\circ}/s$) to “455” (i.e., $+300^{\circ}/s$), with “250” corresponding to “0°/s” (a condition very hard to obtain in practice as the GS-12 was very sensitive to vibrations and signal noises).

In the TASK file “BIO_PR_M_Humanoid_A.tsk”, the function “InitializationGyro” (statements 840–873) collected ten consecutive readings of Ports 3 and 4 and computed their average values “FBBalCenter” and “RLBalCenter” respectively. Next, it used these average values to ascertain whether the GS-12 actually existed or not (statements 861–864). If the GS-12 existed, the parameters “ExistGyro” and “GyroUse” would be set to TRUE, also the “Slip” parameter would be set to “0” (statements 860, 871 and 872). The “Slip” parameter was used to detect whether a “fall” had occurred to the robot (discussed later in Sect. 9.2.2).

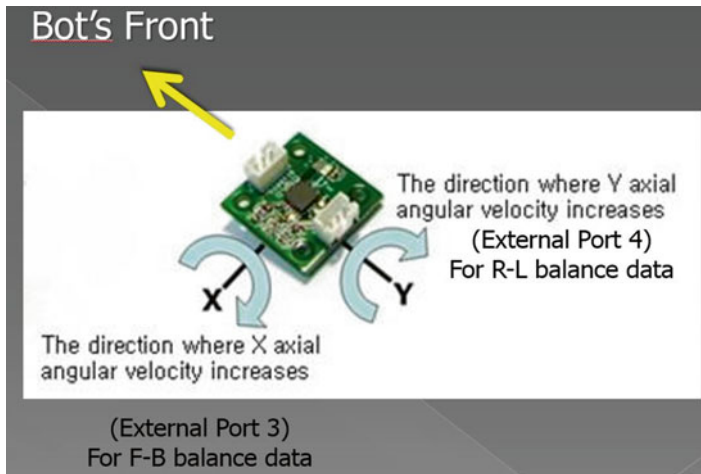


Fig. 9.8 Coordinates system used by the GS-12 for Bal'Act robot

9.2.1 Walk Enhancement with GS-12

If the GS-12 sensor was properly installed and working (parameters “ExistGyro” and “GyroUse” set to TRUE), the CALLBACK function inside the example code “BIO_PRH_Humanoid_A.tsk” was designed to stabilize the FB and RL motions when the robot was performing its programmed actions.

This CALLBACK function illustrated a similar data processing approach to the one used for the AX-S20, but with some important differences:

1. Read in current data from Ports 3 and 4 and compute the error signals “FBBalError” and “RLBalError” (statements 879–883).
2. Scale these error values appropriately and apply these scaled values to the joint offsets of the servos corresponding to the “hip” (IDs=9, 10), “knee” (IDs=13, 14) and “ankle” (IDs=15–18) parts of the robot (statements 890–907).
3. The reader could still see the “mirror” characteristic in setting the joint-offsets of the “left” and “right” matching servos (for example, statements 899 and 901 for the “knee” servos).
4. Importantly, the reader should note that FB error signals were applied only to servos affecting the FB rotational direction, i.e., servos 13, 14, 15 and 16 (statements 899–902). While the RL error signals were applied only to those servos affecting the RL rotational direction, i.e., servos 9, 10, 17 and 18 (statements 904–907). *Please contrast this approach to the one used for the AX-S20 which incorporated both FB and RL error terms in computing the servos joint-offset values (Sect. 9.1.1).*

9.2.2 Fall Detection with GS-12

The CALLBACK function inside the “BIO_PRM_Humanoid_A.tsk” program was also designed to detect a “fall” of the robot via the parameter “FBBalError” (statements 885–888):

1. If the robot fell forward, its FBBalData’s value would be close to 45 (see Fig. 9.8), i.e., smaller than the value for FBBalCenter (which should be around 250) (statement 882). Thus FBBalError’s value would be less than “-200”, and consequently parameter “Slip” would be set to “1” (statements 887–888).
2. Conversely, if the robot fell backward, its FBBalData’s value would be close to 455, thus FBBalError’s value would be more than “200”, and consequently parameter “Slip” would be set to “-1” (statements 885–886).
3. Once the controller finished its CALLBACK cycle and got back to the main code, an IF structure (statements 307–319) would detect this condition and acted on it by first stopping all walking motions (statements 309–310), and then played either Motion Page 27 or 28 depending on whether the robot fell forward (Slip==1) or backward (Slip==-1), respectively. Once the robot finished with either Motion Page 27 or 28, parameter Slip was reset to 0 and the controller continued on with this TSK program. Please note that there was no way for the robot to check if it actually stood back up successfully into its normal standing position, after playing Motion Pages 27 or 28, without the use of a sensor equivalent to the AX-S20.

9.3 Humanoid Balance with FPS

As previously mentioned in Chap. 3, the foot pressure sensor FPS from HUV Robotics is no longer available commercially, but it illustrates an important class of sensor needed for bipedal motion, so it still has some instructional value. This work was also done by Mr. Matthew Paulishen.

The HUV-FPS sensor is Dynamixel-compliant and it actually had four sensing pads, thus acquiring data from them presented a small challenge as the CALLBACK function would only allow two device calls in its code section. Section 9.3.1 showed how to adapt this constraint to the FPS sensor.

9.3.1 FPS Data Acquisition

As an illustration of the approach used, the reader is referred to the two files “BalAct_1Leg_FPS.mtn” and “BalAct_RLeg_FPS_DA.tsk” files. Figure 9.9 showed the Dynamixel IDs and addresses scheme used in these demonstration codes.

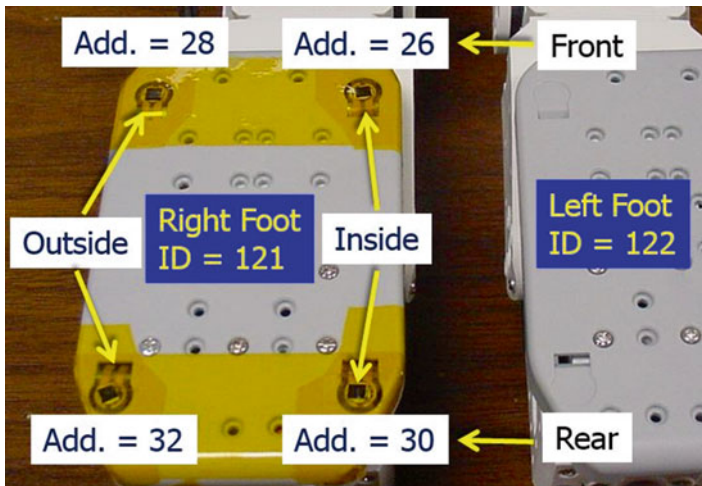


Fig. 9.9 Dynamixel IDs and Addresses scheme for HUV FPS sensor

In the “BalAct_1Leg_FPS.mtn” file, Motion Pages 1 and 2 were used for a right instrumented foot during the “initialization” and “balance” phases respectively. While Motion Pages 5 and 6 were used in case of a left instrumented foot. Motion Pages 2 and 6 were set to get into a continuous play mode.

The “BalAct_RLeg_FPS_DA.tsk” program started out by making the robot go through Motion Pages 1 then 2, i.e., to put the robot into a similar posture as in Fig. 9.6, but on its right leg (see video file Video 9.4). Next, if the user pushed the “Up” button, the bot would collect data from the four pressure pads (addresses 26, 28, 30 and 32) and average them out into four parameters:

1. IFFSR (address 26) for the Inside-Front FSR.
2. OFFSR (address 28) for the Outside-Front FSR.
3. IRFSR (address 30) for the Inside-Rear FSR.
4. ORFSR (address 32) for the Outside-Rear FSR.
5. These initial values were then displayed in the Output Window (statements 57–60).

Then, if the user also pushed the “Right” button, the CALLBACK function would be activated. As there was a limit of two device calls in each 8 ms cycle of the CALLBACK function (statements 80–81), the data collecting and processing scheme used was to rotate through the relevant addresses (26>>28>>30>>32), via the setting of parameter “CurAdd”, one at a time, during each consecutive cycle (statements 84–103). Thus during each CALLBACK cycle, two values of the “current” FSR data were collected (statements 80–81) and its average value saved into the appropriate parameters IFFSR, OFFSR, IRFSR or ORFSR as controlled by the value of “CurAdd” parameter via the IF-ELSE-IF structure.

9.3.2 *Humanoid 1-Leg Balance with FPS*

The “BalAct_1Leg_FPS_Balance.tsk” program was designed to be used with the “BalAct_1Leg_FPS.mtn” file to achieve a 1-Leg balance solution for a Humanoid A robot, using the HUV FPS sensors instead of the AX-S20 as previously done in Sect. 9.1.2.

This TSK program started out by initializing parameters UseBalance and HUMANOID and by zeroing out the joint-offsets of all 18 servos (statements 19–21).

Next, the robot needed to detect how many feet were instrumented and to help the user in deciding which one to serve as the balance foot. It accomplished this task by reading from address 0 of each of the 2 possible Dynamixel IDs (121 and 122, via statements 24 and 25). Then the robot went through an IF-ELSE-IF structure to figure out four possible outcomes (statements 26–74):

1. If no FPS was detected, the robot would play Melody 16 twenty times and quit the TSK program entirely (statements 26–36).
2. If the right FPS was detected, the robot would set parameter FOOT to 1 and play Melody 0 (statements 38–45).
3. If the left FPS was detected, the robot would set parameter FOOT to 5 and play Melody 1 (statements 46–53).
4. If both FPS were detected, the robot would play Melody 0 and then waited on the user to enter his or her choice of either the Left or Right leg to balance on, via the Left or Right buttons respectively (statements 61–73).

The robot next settled into its initial Motion Page (1 or 5 depending on the balance foot—statements 77–81) and waited for the user to push button “Up” (statement 83). And once “Up” was pushed, the robot activated its Balance Motion Page (2 or 6) and called the function “NOIBAL” (statements 109–279).

The function “NOIBAL” was essentially an endless loop wherein the robot could execute several possible tasks:

1. The user could push the “Down” button to make the robot stop balancing (i.e., set UseBalance to FALSE), and to reset all joint offsets by calling the function ZeroJoints (statements 268–276). Then the user could push the “Up” button to set UseBalance to TRUE and call for a new set of reference values for the FSR by calling function ReferenceFootSensorValues (statements 113–119).
2. When UseBalance was set to TRUE, the robot enter a large code segment (statements 120–277), whereas it computed the average value of five FSR readings and compared them to the respective reference values (i.e., “nom” parameters) in order to figure out how much to adjust the joint-offsets of appropriate servos among the servos 11, 12 and 15 for FB balance, and the servos 9, 10 and 17 for LR balance.

The video file Video 9.5 showed the actual performance of this solution.

9.4 HaViMo2 Applications

The HaViMo2 color video camera (Fig. 9.10) was developed by Dr. Hamid Mobalegh for RoboCup applications (http://www.havisys.com/?page_id=8). It was Dynamixel compliant (3-pin TTL) and had a locked ID=100, which happened to be the default ID for a typical AX-S1 sensor also, thus the user would need to make proper ID adjustment on the AX-S1 if he or she planned to use these two sensors on the same robot.

This camera was compatible with the CM-5/CM-510/CM-700/CM-530 controllers via RoboPlus Task or directly via ROBOTIS's Embedded C tools (see Chap. 10). It was known that this camera had some latency issue with the CM-530 firmware (V. 1.1969) and users would need to install an alternate CM-530 firmware (<http://www.havimo.com/?p=130>).

It also worked with the Open-CM IDE on the CM-9.00 and CM-9.04 controllers (see Chap. 10).

9.4.1 HaViMo2 Features and Usage

The HaViMo2 camera was based on the CMOS image sensor HV7131RP (MagnaChip Semiconductor, c. 2005) and had on-chip image processing capabilities (http://www.havimo.com/?page_id=32).

Fig. 9.10 Dynamixel-compliant color camera HaViMo2



The following RoboSavvy.com web site had practical documentation and software downloads for this camera (http://robosavvy.com/store/product_info.php/manufacturers_id/21/products_id/639). Its video frame resolution was at 160×120 pixels, with color depth at 12 bits YCrCb and it had a maximum frame rate of 19 fps (interlaced). Its capabilities were better realized on a faster MCU such as the ARM controller used in the CM-530 and Open-CM-9.00/9.04. This camera required a PC-side application called HaViMoGUI (downloadable from RoboSavvy) to get itself calibrated to the light source used (a very important step) and to set its color look-up table LUT (up to eight distinct colors, i.e., background + 7 user-created). Once the LUT was set, up to 15 contiguous objects (regions) could be identified during run-time. For each of these regions, the camera could report on its Color, row-column coordinates of its Center of Mass, Number of Pixels and its Bounding Box. The user is recommended to consult the enclosed file “HaViMo2UserManual.pdf” for more details Mobagleh (2010).

9.4.2 HaViMo2 Application to a CM-5 CarBot

Figure 9.11 showed the CM-5 robot platform used to illustrate how to interface and program a HaViMo2 camera using the RoboPlus software suite.

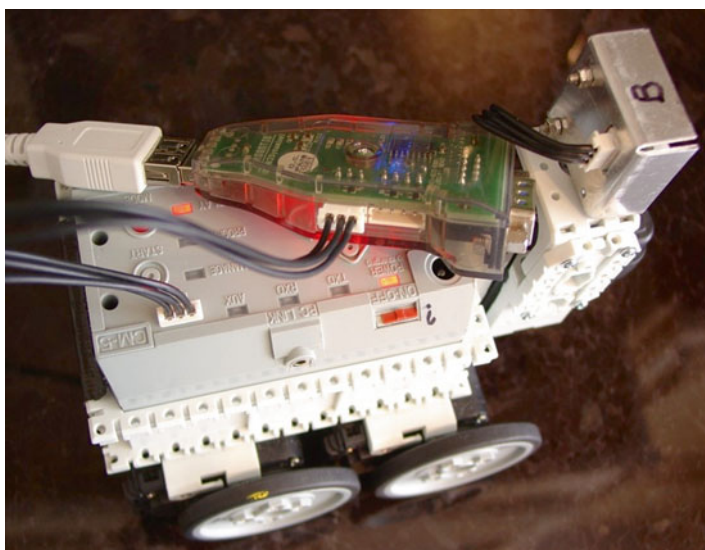


Fig. 9.11 HaViMo2 camera with CM-5 and USB2Dynamixel

As the HaViMo2 camera required rather lengthy procedures for its calibration and programming tasks, two video tutorials were made to inform the reader about those procedures:

1. In “Video 9.6”, the goal was to show how to access the camera via an USB2Dynamixel module (a more direct approach preferred by the author over other “through-connections” via the controllers CM-5XX). The reader is referred to page 3 of the document “HaViMo2UserManual.pdf” for more details on other possible hardware setups. This video also showed how to use the HaViMoGUI application to calibrate the camera and define the LUT to characterize the colors of four painted wooden dowels. Additionally, it illustrated the use of the program “Carbot_Visual_Track.tsk” to track those same wooden dowels.
2. In “Video 9.7”, more fine-tuning of the controlling parameters inside the program “Carbot_Visual_Track.tsk” was performed, and the program “Carbot_Find&Approach.tsk” was demonstrated. Essentially, the Carbot was programmed to find a user-designated colored dowel and drove up to get closer to it.

The reader readily noticed that the color tracking work was performed rather slowly, this was because the CM-5 was an Atmel AVR MCU running only at 16 MHz and we were using the TASK tool which had some computing overhead. When the HaViMo2 was used on a pan-tilt apparatus using a CM-530 and a TASK program, the performance was improved perceptively (https://www.youtube.com/watch?v=pog2gzpjo7g&list=UUGIds85x7Q_nBOReZ818LJQ) because of a faster clock rate and a more efficient MCU were used. When switching to an embedded C version on a CM-510, the performance got further improved showing the power of direct C control of the Atmel AVR MCU (https://www.youtube.com/watch?v=pMbSqkshNZo&list=UUGIds85x7Q_nBOReZ818LJQ&index=35). Finally, on an OpenCM-9.04B, the performance was much more improved, combining faster MCU clock rate and efficient architecture (https://www.youtube.com/watch?v=kCH8F4lXXZM&list=UUGIds85x7Q_nBOReZ818LJQ).

As the tutorial videos might not have enough resolution to allow the readers to see individual lines of codes, the author would like to revisit some of the more important image capture and processing concepts shown in the TSK programs “Carbot_Visual_Track.tsk” and “Carbot_Find&Approach.tsk” (both created by Mr. Matthew Paulishen), in a more text-based manner:

1. **“Carbot_Visual_Track.tsk”**. The most important statement in this TSK program was Line 21 as shown in Fig. 9.12. It was used to capture a new video frame and to start the process of finding the regions as defined by the “colors” chosen by the user in the calibration process with the HaViMoGUI tool (see video “Video 9.6”).

Parameter CamID was hardware-set to 100 and Address **0** was used to trigger the frame capture process, although the function recommended in the “HaViMo2UserManual.pdf” was “CAP_REGION” (i.e., Address **0x0E**) as shown in Table 1 of Page 6 of that PDF document. Next was a “do-thing” loop

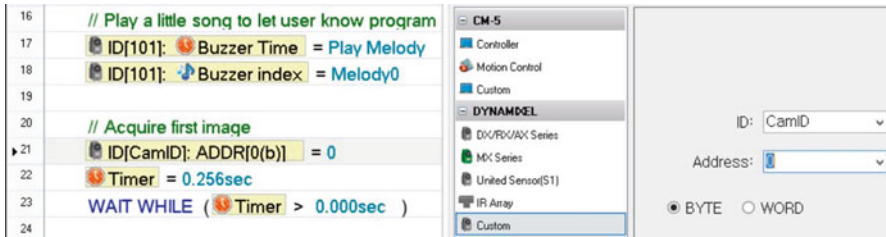


Fig. 9.12 Line 21—CUSTOM Command to capture and process a new video frame with HaViMo2

for 0.256 s (lines 22–23) to allow the first image capture process to go through before the TSK program went into its Endless Loop to find the wanted color object and track it with the Carbot’s AX-12s.

Line 28 called the function “Get_Bounding_Box” (defined at lines 81–113) which looked at the data structure returned by the HaViMo2 to see if any valid region, defined by the user’s “Color” parameter as set on Line 14 of this TSK program, was actually found. Figure 9.13 displayed the FOR loop used to go through the 15 possible regions provided by the HaViMo2 firmware for each frame captured and processed (see line 84). Thus, for example, “Index” equaled “1” for the first region and “Addr” correspondingly equaled “16”. Next the TSK program looked at the “Byte” found at Address “Addr” (line 87), and if this “Byte” was non-zero, this was a valid region. Essentially, this “Byte” corresponded to the parameter “Index” as shown in Fig. 3 of the “HaViMo2UserManual.pdf” document.

The information in this Fig. 3 was also reproduced below for the reader’s convenience:

- Index (1 byte)—zero if the region is invalid and nonzero otherwise.
- Color (1 byte)—color code of the detected region (0=Unknown, or 1–7).
- Pixels (2 bytes)—number of detected pixels inside the region.
- SumX (4 bytes)—sum of the X coordinates of the detected pixels.
- SumY (4 bytes)—sum of the Y coordinates of the detected pixels.
- MaxX (1 byte)—bounding box right margin.
- MinX (1 byte)—bounding box left margin.
- MaxY (1 byte)—bounding box bottom margin.
- MinY (1 byte)—bounding box top margin.

Now that the TSK program had read in the value for parameter Index (line 87) and found that Index was non-zero, it would next increment Addr by 1 to get to the Color member of this 16-byte data structure. If this Color member matched with the Color parameter (set back in line 14), it incremented Addr again by 1 to get to the Size of the current detected region (2 bytes = 1 word, see lines 92–93). In this particular example, we were interested in tracking only ONE object/region, thus the use of the IF structure to find the biggest one among the 15 regions (see

```

81 FUNCTION Get_Bounding_Box
82 {
83     Max = 0
84     LOOP FOR ( Index = 1 ~ 15 )
85     {
86         Addr = Index * 16
87         IF ( ID[CamID]: ADDR[Addr(b)] != 0 )
88         {
89             Addr = Addr + 1
90             IF ( ID[CamID]: ADDR[Addr(b)] == Color )
91             {
92                 Addr = Addr + 1
93                 Size = ID[CamID]: ADDR[Addr(w)]
94                 IF ( Size > Max )
95                 {
96                     Max = Size
97                     MaxAddr = Addr
98                 }
99             }
100         }
101     }

```

Fig. 9.13 Function “Get_Bounding_Box” to extract data out of the 15 regions

lines 94–98). At the completion of the FOR LOOP, parameter Max would contain the Size of the largest valid region with the matching Color, and parameter MaxAddr would contain the current pointer address (i.e., still pointing at the Pixels data member of the 16 byte data structure for this largest valid region).

Next, if Max was non-zero (i.e., valid region), the TSK program started from the MaxAddr value and jumped ahead by 10 bytes to get to the MaxX data member and saved that value in parameter Maxx (lines 104–105). Then it incremented Addr by 1 byte to get to MinX and therefore Minx (lines 106–107). Parameters Maxx and Minx would be used later in the main function to compute the appropriate steering commands for the CarBot in order to keep the user-defined object in front of the HaViMo2 camera as much as possible (lines 29–73).

2. **“Carbot_Find&Approach.tsk”**. This TSK program used the same function “Get_Bounding_Box” to find the largest region of interest, but this implementation used the RC-100’s buttons 1-2-3-4 to set the color wanted by the user on the fly (lines 31–61). It also added new codes to check on the current Size of the target (i.e., region of interest) and from there it could decide whether to com-

mand the carbot to get closer or to back away from the target to maintain a user-given TargetSize (lines 103–126).

9.5 Review Questions for Chap. 9

1. Why was the AX-S20 sensor mounted in the head of the Humanoid robot?
2. Why was the GS-12 sensor mounted in the abdomen of the Humanoid robot?
3. How many device calls are allowed in the CALLBACK function?
4. Is there a size limitation on the CALLBACK function?
5. What was the resolution in degree of the Azimuth, Pitch and Roll angles provided by the AX-S20?
6. How many bits were contained in the X-Y-Z acceleration data provided by the AX-S20?
7. What is the refresh rate for data provided by the AX-S20 sensor?
8. What is the refresh rate for data provided by the GS-12 sensor?
9. What was the procedure used to activate the Joint Offsets of the actuators used in the balancing of the Humanoid robot whether one uses the AX-S20 or GS-12?
10. The AX-S20 is a Dynamixel-compliant sensor. (T-F)
11. The GS-12 is a Dynamixel-compliant sensor. (T-F)
12. What is the value range for the digital output from the GS-12 sensor?
13. The HUV Robotics FPS sensor is a Dynamixel-compliant sensor. (T-F)
14. The HaViMo2 camera is a Dynamixel-compliant sensor. (T-F)
15. What is the pixel resolution of the HaViMo2 camera?
16. What is the maximum video frame rate for the HaViMo2 camera?
17. What is the color space used by the HaViMo2 camera? What is its color depth in terms of binary bits?
18. How many distinct colors can be saved in the color look-up table for the HaViMo2 camera?
19. How many contiguous objects/regions can the HaViMo2 search for during each cycle of operation?

9.6 Review Exercises for Chap. 9

1. Starting from the program “Carbot_Find&Approach.tsk”, the reader could develop a slalom negotiating Carbot project using the HaViMo2 camera to weave around colored dowels, as shown in this video clip from a University of Georgia student (see video file “Video 9.8”).
2. Other National Taiwan University students mounted the HaViMo2 camera onto a PREMIUM GERWALK to perform the same Find and Approach feature for a

given color patch. This work was more challenging as the GERWALK was always “jiggling” the camera, thus it could only capture and process images during a short time when the camera was level. The students also used the DMS sensor to command the robot to stop when close enough to the target (view file “Video 9.9” or <https://www.youtube.com/watch?v=HWxMwvFriMc>).

References

- Chevallereau C et al (2009) Bipedal robots. Wiley, Hoboken
Kajita S et al (2014) Introduction to Humanoid robotics. Springer, Heidelberg
Mobagleh H (2010) HaViMo2 image processing module. <http://robosavvy.com/RoboSavvyPages/Support/Hamid/HaViMo2.pdf>. Accessed 29 Dec 2014

Chapter 10

Embedded C Options

Chapter 9 showed that in order to get the maximum performance out of ROBOTIS systems, ones must consider the “Embedded C” routes which happened to be quite numerous. Some of the resources that I know of are listed below (and I know that I have missed many):

- Vanadium Labs had been supporting the ArbotiX RoboController line, Atmel AVR-based (<http://www.vanadiumlabs.com/arbotix.html>). It is supported by the ROS organization (<http://wiki.ros.org/arbotix>).
- The Humanoid Lab provides C libraries for the Bioloid Premium (<http://apollo.upc.es/humanoide/trac/wiki/bioloidCframeworks>) via Windows and Linux.
- BioloidCControl is an alternative firmware for BIOLOID PREMIUM Humanoids A/B/C (<https://code.google.com/p/bioloidccontrol/>).
- “Software Souls” also offer another approach (<http://softwaresouls.com/softwaresouls/series/programming-robotis-bioloid-hardware/>).
- Another interesting product is the USB2AX which is equivalent to the USB2Dynamixel module (<http://www.xevelabs.com/doku.php?id=product:usb2ax:usb2ax>) for Windows, Linux and MacOS systems.
- As far as Arduino books go, there are many in existence to suit the reader’s skills and goals, but I would recommend the classic “Arduino Cookbook” by Margolis (2011).

However this chapter’s main topics would stay with the official ROBOTIS Embedded C routes. Historically speaking, Embedded C was first available for the CM-5 via its BIOLOID Expert Kit (c. 2006, but it is no longer available) and currently Embedded C functionality is offered only for the CM-510/700/530 and the OpenCM-9.00/9.04 series.

Electronic supplementary material: The online version of this chapter (doi:[10.1007/978-3-319-20418-5_10](https://doi.org/10.1007/978-3-319-20418-5_10)) contains supplementary material, which is available to authorized users.

Chronologically and for the international market, the Embedded C utilities for the CM-510 came out first in early 2010, next were the ones for the CM-530 in Spring 2012. While the OpenCM-9.00/9.04 systems came out in 2012–2013 and the last hardware component OpenCM-485-EXP came out in Summer 2014. However, for a better flow of the topics in this book, I would start with the OpenCM IDE for the OpenCM-9.00/9.04 series and end this chapter with Embedded C for the CM-510/530 systems, as I expect that most readers of this book would start with the RoboPlus system and the CM-5XX series for their ROBOTIS journey. I also realized that I would have to assume that the reader had some prior knowledge of C/C++ programming or had access to a good C/C++ programming resource, as that topic would be outside the scope of this book. Thus my overall goal for this chapter was to contrast similarities and differences in usage between RoboPlus and Embedded C tools regarding:

- General use differences to note between RoboPlus TASK and Embedded C.
- OpenCM IDE for the OpenCM-9.00/9.04 family.
- Embedded C options for the CM-510 and CM-530 systems.
- Motion Programming and Embedded C.

10.1 Embedded C vs. RoboPlus' TASK

The TASK tool was designed for beginners in robotics and also in computer programming skills, thus its interface shielded the user from important details that could not be ignored when switching to Embedded C interfaces:

1. **Data Types.** A TASK programmer only needed to declare a parameter's name and started to use it anywhere in the TSK code, and this parameter could only handle integer values anyhow. Embedded C would allow all the standard data types—integer, floating-point and more complex data structures. The TASK programmer could already get a taste of things to come with the CUSTOM command for Controller and Dynamixel (see Figs. 9.12 and 9.13) whereas one had to choose between a BYTE (8 bits) or a WORD (16 bits) data type to properly read or write to a given parameter or device (such as for the HaViMo2 camera in Chap. 9).
2. **Assignments and Function (Method) Calls.** Typically, about 90 % of all the statements used in a TASK program would be of the Assignment type ($A=B$, i.e., the value of parameter B is assigned to parameter A). For example, to set a Goal Position on an AX-12 with ID=3 to a value of 800, the TASK user would type in the following statement:

ID[3]: Goal Position = 800

Switching to Embedded C, the C/C++ user would use a Function (Method) Call instead:

Dxl.writeWord(3, 30, 800);

where argument “3” corresponded to the ID, while argument “30” corresponded to the address of the Goal Position parameter as defined in the Control Table of the AX-12 (see web link at http://support.robotis.com/en/product/dynamixel/ax_series/dxl_ax_actuator.htm), and argument “800” was the desired value for the Goal Position. In other words, the majority of the statements used in Embedded C would be Method Calls which required the proper setting of many formal parameters to properly use that Method. Therefore it would require the user to be familiar with the Control Table of each type of ROBOTIS actuators and sensors that were used in the robot being considered. This Control Table would also inform the user of the proper Data Type to use (BYTE or WORD).

An alternative Method Call, with a syntax much closer to the one used in the previous TASK statement, could also be used:

Dxl.goalPosition(3, 800);

3. **Devices.** In the previous sub-section, the reader might have noticed the “Dxl.” notation used in the Method Call `Dxl.goalPosition(3, 800)`. Actually, there were two other statements that must have been asserted before a C/C++ programmer could use the Method `goalPosition()` properly:

- (a) **Dynamixel Dxl(1);** to define the object “Dxl” as a “Dynamixel” device that was associated with Serial Bus “1” (i.e., argument “1” within the parentheses) which corresponded to the 3-pin TTL bus on each ROBOTIS controller (from CM-5 to OpenCM-9.04—see Fig. 10.1).
- (b) **Dxl.begin(3);** to initialize the communication with the object “Dxl” (i.e., with the Dynamixel 3-pin TTL bus) at a baud rate of 1 Mbps (i.e., argument “3” within the parentheses).

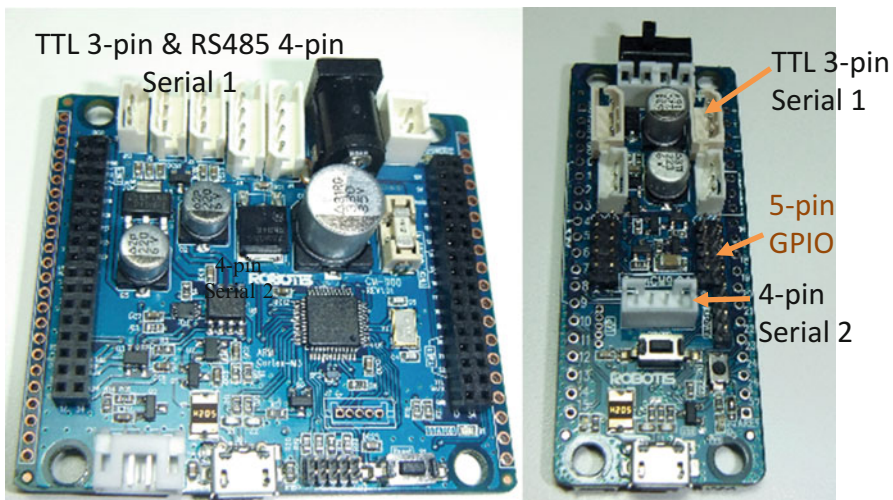
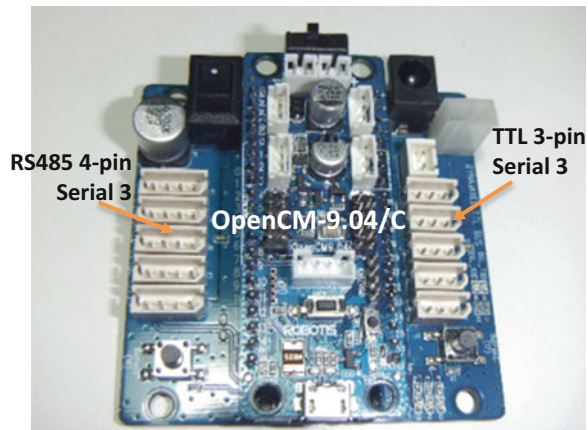


Fig. 10.1 OpenCM-9.00V. 1 (left) and OpenCM-9.04/B (right)

Fig. 10.2 OpenCM--9.04/C on *top* on a 485 EXP shield



For devices using the 5-pin bus (see Fig. 10.2), a similar process needed to be used, but with the OLLO device:

- (a) **OLLO myOLLO;** to define the object “myOLLO” as an “OLLO” device corresponding to the GPIO 5-pin bus on ROBOTIS controllers such as CM-510/530 and OpenCM-9.04/A/B/C.
- (b) **myOLLO.begin(2);** to initialize the particular “myOLLO” device that was hooked up to the GPIO port “2” (as an example).
- (c) **myOLLO.read(2);** to read the device’s current response at port “2” as a digital value.

Please note that a TASK programmer would never have to worry about this kind of details, as the TASK tool took care of associating and initializing Dynamixel and OLLO devices in the background. The video file “Video 10.1” was a side-by-side comparison of a TSK program (on a CM-530—“TSKvsIDE.tsk”) and a C program (on an OpenCM-9.04/B—“TSKvsIDE.ino”) designed to do the same operations on an AX-12 actuator and a DMS sensor. This video file also showed how to adapt this “TSKvsIDE.ino” sketch to the situation when the AX-12 was connected to the TTL bus on the 485-EXP board (see next paragraph on communication ports for more details).

4. **Communication Ports.** Just as “Dynamixel” devices were connected via Serial Bus 1, per-se “communication” devices such as the LN-101 (wired) and ZIG-110 or BT-110/210 (wireless) were programmed to connect via Serial Bus 2 (i.e., through the 4-pin connector (see Figs. 10.1 and 10.2). At present (i.e., November 2014), only the OpenCM 485-EXP expansion shield is assigned to Serial Bus 3 (see Fig. 10.2) and it can be programmed with the OpenCM IDE but NOT with the TASK tool.

5. **Dynamixel Communication Protocols 1 and 2.** Previously, all ROBOTIS Dynamixels used the same communication protocol (http://support.robotis.com/en/product/dynamixel/dxl_communication.htm), but with the introduction of the PRO line in 2012 and then the XL-320 in 2013, a second communication protocol was needed with different instruction/status packet design and faster baudrates (http://support.robotis.com/en/product/dynamixel_pro/communication.htm). At present, ROBOTIS supports mixed-protocol programming only on the OpenCM IDE (i.e., with the OpenCM-9.04/A/B/C) and it would require the use of the Method `setPacketType` (DXL_PACKET_TYPE) whereas the parameter DXL_PACKET_TYPE is set (as expected) to “1” for Protocol 1 and “2” for Protocol 2. Most importantly, the Method `setPacketType()` had to be used to set the appropriate DXL_PACKET_TYPE **before** any communication to the respective type of Dynamixel (protocol 1 or 2)—see example code below:

```
// Dynamixel with ID_1 uses Protocol 1
Dxl.setPacketType(1);
Dxl.goalPosition(ID_1, 512); // go to position 512
delay(1000); // delay 1 second
// Dynamixel with ID_3 uses Protocol 2
Dxl.setPacketType(2);
Dxl.goalPosition(ID_3, 1023);
delay(500); // delay 0.5 second
```

10.2 Embedded C for the OpenCM-9.00/9.04

The current version for the OpenCM IDE is version 1.02, available since early 2014 (http://support.robotis.com/en/software/robotis_opencm.htm) and a User’s Manual for the OpenCM-9.04 system and other hardware information are available at <http://support.robotis.com/en/product/controller/opencm9.04.htm>. Although the OpenCM-9.00 controller is no longer available commercially, enclosed with this book is a ZIP file that has technical information for this system (OpenCM-900-Manuals.zip). Other “community” resources exist such as “robotsource.org”, “trossenrobotics.com” and “robosavvy.com”, just to name a few. Also as the OpenCM IDE was based on the Arduino architecture, there are numerous resources in print and on the web for Arduino that the user could consult to go beyond the ROBOTIS manuals.

As the ROBOTIS manuals and web sites were already providing good instructions for installing the OpenCM IDE and for its basic and advanced uses, this section would only illustrate two example projects that hopefully will be useful to the beginning Embedded C programmer:

1. The video file “Video 10.2” described the remote control of a CarBot equipped with BT-210 by the Virtual RC-100 from inside the ROBOPLUS MANAGER

tool on the PC side. The corresponding Arduino Sketch file was named “CM9_Carbot_RC.ino” and included with this book.

2. The video file “Video 10.3” described the use of a 485-EXP expansion shield with an OpenCM-9.04/B and also the mixing of Dynamixel protocols. The corresponding Arduino Sketch file was named “CM9_Mixed_Protocols.ino” and included with this book.

In Sect. 9.4.2, an early model of OpenCM-9.04/B was shown to have worked well with the HaViMo2 camera using the ROBOTIS IDE V.1 (c. 2013) (https://www.youtube.com/watch?v=kCH8F4lXXZM&list=UUGIds85x7Q_nB0Re-Z818LJQ). Unfortunately, the author had been not able to reproduce the same results using the current model of the OpenCM-9.04/B and the current ROBOTIS IDE V.1.0.2. The attempted sketch “CM9_CarbotHaViMo2.ino” was enclosed with this book for the reader’s reference. This sketch compiled successfully but the HaViMo2 never responded to the first ping to this Dynamixel, invoked in a call to the method `hvm2.ready()`.

10.3 Embedded C for the CM-510 and CM-530

The current C/C++ programming SDK (V. 1.02) for the CM-510/CM-700 is accessible at http://support.robotis.com/en/software/embedded_c/cm510_cm700.htm while the corresponding one for the CM-530 can be found at http://support.robotis.com/en/software/embedded_c/cm530.htm.

These two web sites also have detailed information about installing the needed tool chains for these two controllers:

1. The CM-510/CM-700 tool chain recommended by ROBOTIS is WinAVR and Atmel Studio (http://support.robotis.com/en/software/embedded_c/cm510_cm700/embedded_c_start.htm). Although it is also possible to use the Eclipse IDE with WinAVR (see Sect. 10.3.1).
2. The CM-530 tool chain recommended by ROBOTIS is JRE, WinARM and Eclipse (http://support.robotis.com/en/software/embedded_c/cm530/embedded_c_start_stm.htm).

10.3.1 Tutorials for CM-510

ROBOTIS provided tutorial information for the programming of the CM-510 at (http://support.robotis.com/en/software/embedded_c/cm510_cm700/programming.htm) using the Atmel Studio tool. ROBOTIS also provided many worked out examples at (http://support.robotis.com/en/software/embedded_c/cm510_cm700/example.htm).

If the reader is more interested in using Eclipse with the CM-510, the reader needs to review the series of 4 video tutorials created by Dr. Yanfu Kuo from National Taiwan University on YouTube:

1. <https://www.youtube.com/watch?v=csgotzBhbmI>—Tutorial 1 described the basic architecture of the Atmel AVR ATmega2561 and showed how it was implemented on the CM-510. It also showed the detailed steps for programming the CM-510 from creating an Eclipse project, coding and downloading to the controller using the ROBOTIS Terminal tool. Tutorial 1 showed how to control devices such as LEDs, buttons, serial communication, buzzer and microphone already built-in on the CM-510 controller.
2. <https://www.youtube.com/watch?v=OIUkl6iBUfM>—Tutorial 2 dealt with sensor interfacing issues using the OLLO NIR sensor as the example. It continued on showing how to control Dynamixel actuators such as the AX-12, and ended with a ZigBee communications programming example.
3. <https://www.youtube.com/watch?v=6jZtmN3PXey>—Tutorial 3 showed how to program the HaViMo2 video camera to track colored objects using a pan-tilt platform constructed with two AX-12 actuators. The source code is enclosed in the ZIP file “HaViMo2.zip”.
4. <https://www.youtube.com/watch?v=GGjeCOuuu9M>—Tutorial 4 demonstrated how to interface the Gyro sensor on a CarBot platform and how to integrate the Gyro’s angular rate data to obtain an estimate of how much of an angle (in degrees) that the CarBot had rotated after a given maneuver. The source code is enclosed in the ZIP file “GyroCompass.zip”.

10.3.2 Tutorials for CM-530

ROBOTIS provided tutorial information for the programming of the CM-530 at (http://support.robotis.com/en/software/embedded_c/cm530/programming_stm.htm) using the Eclipse-WinARM tool chain. ROBOTIS also provided several programming examples at (http://support.robotis.com/en/software/embedded_c/cm530/example_stm.htm).

Unfortunately, there no video resources, known to the author, showing how to use of the Eclipse-WinARM tool chain for the CM-530, but the CM-510 tutorial series provided by Dr. Yan-Fu Kuo should serve as a good starting point as they were also using Eclipse.

10.3.3 Future Support for Embedded C for CM-510/530?

At present, ROBOTIS had not been updating the CM-510/530 libraries for the new sensors such as the IR Sensor Array, Color and Magnetic sensors, nor for the 485-EXP Expansion module, but they did recently release the new Windows and Linux

Dynamixel Communications Protocol 2.02 (http://support.robotis.com/en/software/dynamixel_sdk/usb2dynamixel/usb2dxl_windows.htm#bc-1).

10.4 Motion Programming and Embedded C

With Embedded C SDKs, ROBOTIS had graciously shared their knowledge and expertise for their robotic systems (BIOLOID and OpenCM) with robotics enthusiasts everywhere. However, ROBOTIS still kept their Motion Programming technologies pretty much proprietary, and this move was very understandable by anyone who had a closer look at what Motion V.2 could do (more on this tool in Chap. 11). At present, there are only a few resources dealing with Motion Programming with the OpenCM IDE.

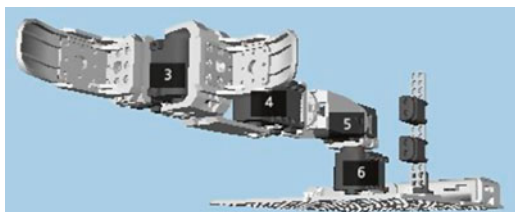
Included with the OpenCM IDE (V.1.0.2) were 2 ROBOTIS examples showing the basics of creating Motion Pages and how to synchronize-play these Motion Pages:

1. The first sketch “q_Motion_Page_Play” is accessible via the “File” pull-down menu»Examples»06.Dynamixel»q_Motion_Page_Play.
2. The second sketch “s_Manipulator_4DOF” is accessible via the “File” pull-down menu»Examples»06.Dynamixel»s_Manipulator_4DOF.

The video file “Video 10.4” demonstrated the operation of the “q_Motion_Page_Play” sketch, modified to work with the RC-100 on a set of 4 AX-12s mounted in a CarBot, but these actuators could also be constructed into a robotic arm such as the one shown in Fig. 10.3.

Mr. Matthew Paulishen had contributed many advances for our UGA robotics laboratory as shown in earlier Chaps. 8 and 9. In 2013, he adapted the PyPose tool from Vanadium Labs (<http://vanadiumlabs.github.io/pypose/>) into the OpenCM IDE as the library CM9_BC which could convert a standard MTN file into PyPose data sets. A presentation of his work can be found on YouTube at <https://www.youtube.com/watch?v=iPD5oOFYq3Y>. The application of the CM9_BC tool to an OpenCM-9.00 V.1 can be viewed at this link <https://www.youtube.com/watch?v=4X-KbiOPE4s>, while an application on the OpenCM-9.04/B can be watched at this web site <https://www.youtube.com/watch?v=EdlTAVn2d3s>.

Fig. 10.3 4-DOF robotic arm from BIOLOID STEM EXPANDED kit



More recently (April 2014), Dr. Hamid Mobalegh created a direct walking engine into an OLLO biped, with four XL-320s, as a sketch for the OpenCM-9.04/B (<http://www.havisys.com/?p=148>).

In conclusion, the Open-CM IDE seemed to be well accepted by a still small community but interested in developing Open Motion Programming capabilities for the OpenCM-9.04 systems.

10.5 Review Questions for Chap. 10

1. Variables in TASK programs can be of the floating-point types. (T-F)
2. An assignment statement in a TASK program would be equivalent to a function call in Embedded C. (T-F)
3. What ROBOTIS database would an Embedded C programmer consult in order to user proper data types for variables used?
4. What is the most important object in Embedded C for ROBOTIS hardware?
5. Which type of data bus is the device type Dynamixel associated to?
6. Which type of data bus is the device type OLLO associated to?
7. How many serial channels does the OpenCM9.04 systems provide?
8. Which serial channel is the shield 485 EXP board associated with?
9. Which serial channel are the GPIO ports associated with?
10. Which serial channel are the Dynamixel ports associated with?
11. Which serial channel is associated with communication devices such as ZigBee and BlueTooth?
12. Which Dynamixel Communication Protocol does an MX-28 actuator respond to?
13. Which Dynamixel Communication Protocol does an XL-320 actuator respond to?
14. Describe the tool chain needed to use Embedded C on the CM-510.
15. Describe the tool chain needed to use Embedded C on the CM-530.
16. Describe the tool chain needed to use Embedded C on the OpenCM-9.04/B.

10.6 Review Exercises for Chap. 10

ROBOTIS had created extensive Embedded C example programs for their controllers CM-510, CM-530 and OpenCM-9.04/A/B/C that the readers can refer to when they install these resources on their own PCs.

Reference

Margolis M (2011) Arduino cookbook. O'Reilly Media, Sebastopol

Chapter 11

ROBOTIS-MINI System

In the Spring of 2014, the DARWIN-MINI system was released internationally using a new controller (OpenCM-9.04-C), with a new communication device (BT-210) and the new RoboPlus 2.0 suite: TASK, R+ DESIGN and R+ MOTION (V.2). So far (March 2015), the RoboPlus Manager tool (V.1.0.33.2) could only serve as a firmware updater to the 9.04-C, but the RoboPlus Dynamixel Wizard tool (currently at V.1.0.19.5) had been modified to work with OpenCM-9.04-C quite well as a Firmware Update/Recovery tool for the XL-320 actuators as well as for the 9.04-C controller (see Sect. 4.2.2). The OpenCM-9.04-C could also be used with the OpenCM IDE V.1.0.2, but it would need a firmware recovery (via Manager only and the micro USB port) to make it work again with the RoboPlus 2.0 suite. In November 2014, the second edition of this system with an improved XL-320 (metal pinion gear) was renamed ROBOTIS-MINI.

The robot programming concepts behind ROBOTIS MOTION tools could be generalized to a research area named Programming by Demonstration (PbD), also known as Imitation Learning or Apprentice Learning. The interested reader is referred to Billard et al. (2008), Calinon (2009) and also these web links:

- http://www.scholarpedia.org/article/Robot_learning_by_demonstration
- <http://programming-by-demonstration.org/>

This chapter's main topics are listed below:

- PC wireless options for the MINI.
- New motion concepts in MOTION V.2.
- PC motion control with TASK and MOTION V.2.
- LED integration with TASK and MOTION V.2.
- Choreography for two MINIs.

Electronic supplementary material: The online version of this chapter (doi:[10.1007/978-3-319-20418-5_11](https://doi.org/10.1007/978-3-319-20418-5_11)) contains supplementary material, which is available to authorized users.

11.1 PC to MINI Wireless Options

On a mobile device, such as an Android tablet, BlueTooth comes standard thus mobile users can only use BlueTooth with the ROBOTIS-MINI. However on the PC, users could choose between ZigBee (ZIG-110A) and BlueTooth (BT-110A or BT-210).

The ZIG-110A by default worked at 57.6 Kbps but it could only go up to 115.2 Kbps as it was an older ROBOTIS product (c. 2009), but it could emulate three modes of communications: 1 to 1 (1:1), 1 to many (1:N) or broadcast (N:N). In the author’s experiences, upon powering up ZigBee usually achieved connections much quicker and more reliably than BlueTooth. On the PC side, the user had to use a combination of three modules (USB2Dynamixel + Zig2Serial + ZIG-100) to make the connection, but it used only one Serial COM port on the PC side. The ZigBee N:N option, once set up properly (see Sects. 8.1 and 8.2), could be particularly convenient if ones needed multiple robots to communicate with each other without involving the PC and this work could be done via the current TASK tool.

The BT-110A (BT specification 2.0) by default was also set to 57.6 Kbps but could reach out to 230.4 Kbps, while the BT-210 (BT specification 2.1) could get up to 400 Kbps. On the PC side, most new PCs would come with a built-in BlueTooth server (specification 4.0 at present) or a small USB device could be purchased to fulfill this role on an older PC. Upon connection to the PC, the PC OS would use two Serial COM ports per BT device (see Fig. 11.1) and the user would have to take care to pick only the OUTGOING COM ports to connect between the various ROBOPLUS software tools (TASK, MANAGER, DYNAMIXEL WIZARD) and multiple MINIs. However, since V.2.2.3, the R+MOTION tool filtered out the INCOMING COM port, and thus presented only the OUTGOING COM port to the user, which was a good step forward.

Fig. 11.1 BT settings on PC with 1 BT-110A and 2 BT-210 connected

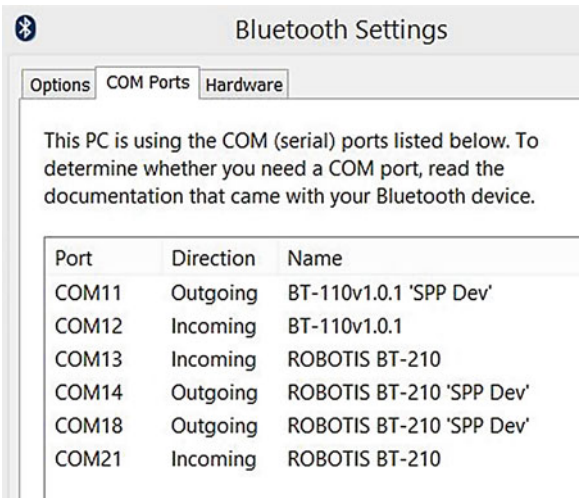


Figure 11.1 also implied that BT communications between multiple robots would have to be mediated via the BT server on the PC, which would require more programming resources and skills beyond the TASK tool. As a matter of fact, ROBOTIS recently released a technical note regarding the pairing of BT-210s using the ROBOTIS IDE and an OpenCM-9.04/B controller (http://support.robotis.com/en/techsupport_eng.htm#product/auxdevice/communication/bt-210_setting.htm).

Summing up, it really depended upon the user's needs, monetary funds and current programming expertise to choose the proper wireless protocol to use with the ROBOTIS-MINI system. Working with only one MINI, the BT-210 would be the most economical way to go for PC and Android platforms.

Around Summer 2015, ROBOTIS plans to release the new BT series BT-410 Master and Slave modules to allow 1:1 and 1:N communications. The BT-410 series would be based on Bluetooth 4.0 Low Energy (see Sect. 3.2.3).

11.2 New Motion Concepts in MOTION V.2

Between Version 1 and Version 2 of the MOTION tool, ROBOTIS had made quite a few fundamental changes such as: file suffix change from MTN to MTNX, removing size limitation on the physical file, changing internal data structures for easier motion design and editing, synchronization between the 3-D simulated robot moves and the actual physical moves of the demonstration robot.

The English version of the user manual for MOTION V.2 is available at (http://support.robotis.com/en/software/roboplus2/r+motion2/rplus_motion2.htm) and has many detailed procedures that the reader should review as needed. The ROBOTIS Development Team also hosted a YouTube channel where the reader could watch more tutorial videos at <https://www.youtube.com/channel/UCuHS2rd-R6LjKyw3yTKXObA/videos>.

11.2.1 *Unlimited File Size for MTNX*

The old MTN motion file had a maximum file size that was linked to the working memory size of the respective CM-5XX controllers, i.e., 127 motion pages for CM-5 and 255 motion pages for the CM-510 and CM-530.

The new MTNX file no longer has an upper limit for its physical size thanks to a new Motion Data structure being implemented (see next Sect. 11.2.2). The MTNX file was also now referred to as a "Project" in ROBOTIS' technical documents.

11.2.2 Efficient Motion Data Structure

Motion data in Version 1 (see Fig. 11.2) could be described hierarchically as:

- 1. POSE—specify a set of user-defined goal position values for all actuators used by the demonstration robot and at an instant in time.
- 2. STEP—specify a set TIME interval for the robot to reach a given POSE. Up to a maximum of seven STEPS could be defined per motion PAGE which could be considered as a small move by the robot. A PAUSE time interval could also be defined for “between” STEPS.
- 3. PAGE—each motion PAGE can be linked to a NEXT page to create more elaborate robot gestures. An EXIT page could also be defined to ensure a stable position for emergency stops. The PAGE NUMBERS defined could be triggered or “played” from a controlling TASK program, resulting in the actual performance of the robot’s moves.

Figure 11.3 was a screen capture of the main Motion Editing interface for MOTION V.2 which used the Unity graphics engine (<http://unity3d.com/>).

MOTION V.2 used a Global Time Line where the smallest Time Frame allowed was 8 ms which corresponded to the refresh cycle time for all ROBOTIS Dynamixels (see Fig. 11.3). This 8 ms timing also explained the synchronization process between the simulation graphics and the real timing of the robot’s executed moves.

Each robot POSE still stood for a set of goal position values of the robot’s actuators which could be manipulated singly or as a group using the POSITIONING tool, located in the lower right corner of Fig. 11.3, with full 3-D graphical feedback on the robot model. Once satisfied with a given POSE, the user could insert it into a wanted time frame on the Global Time Line to make it become a KEY FRAME. Several KEY FRAMES would form a MOTION UNIT (see Fig. 11.4).

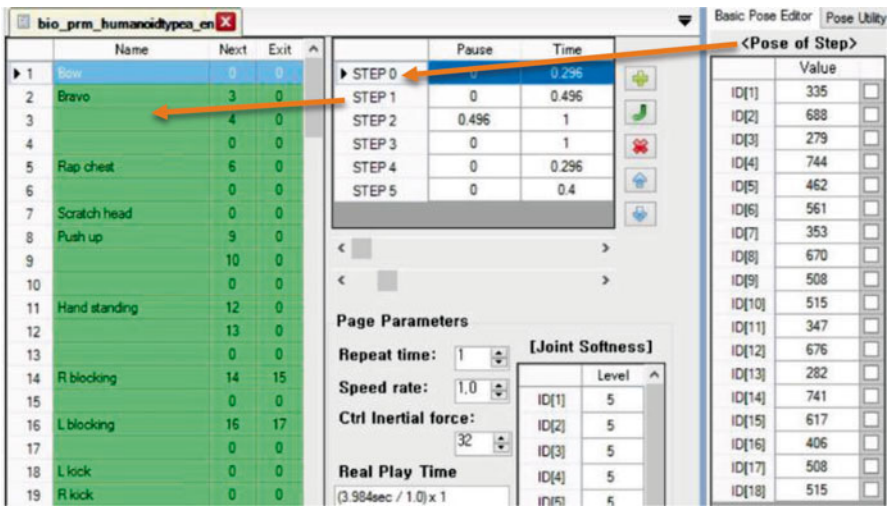


Fig. 11.2 Motion Data Structure used in Motion V.1

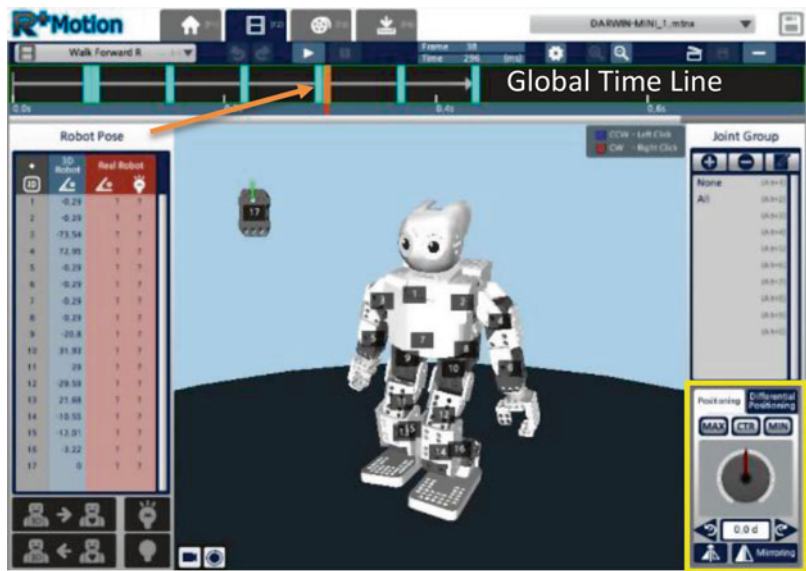


Fig. 11.3 Motion Unit Editing Interface in Motion V.2

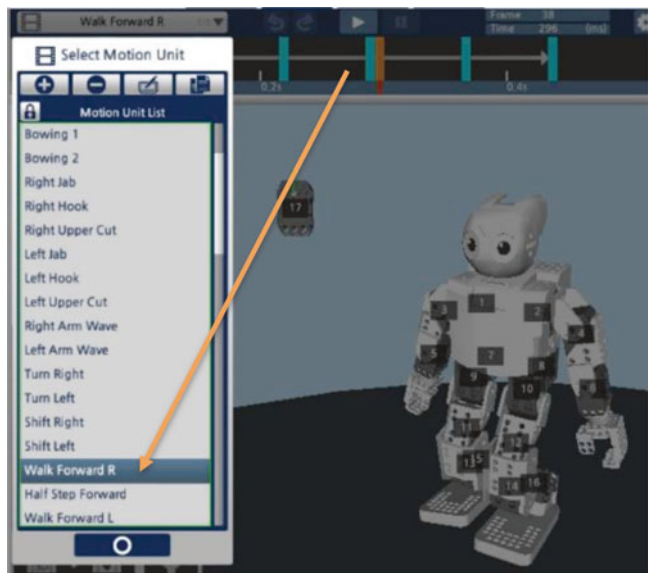


Fig. 11.4 Listing of user-created Motion Units

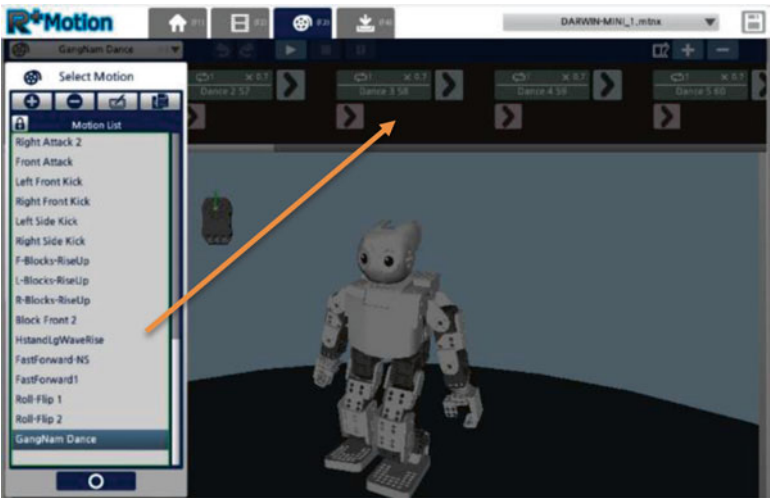


Fig. 11.5 Editing Motion Units into a Motion List



Fig. 11.6 Creating Motion Group with user-selected Motion Lists

Several MOTION UNITS could then be edited into a MOTION LIST which essentially performed the Flow Control task for the selected MOTION UNITS (see Fig. 11.5).

The next step for the user was to create a custom MOTION GROUP which had user-selected MOTION LISTS as “independent” members of this Motion Group (see Fig. 11.6).

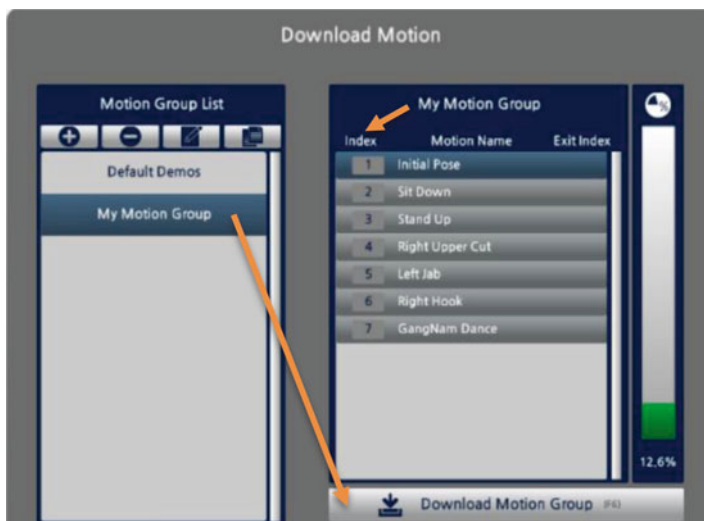


Fig. 11.7 Downloading chosen Motion Group

The user could create several MOTION GROUPS to be saved in the same MTNX file (because its physical size on the PC is now unlimited), but the user could DOWNLOAD only ONE Motion Group to the MINI at any one time, because the working memory on the MINI was still finite (see Fig. 11.7)

The INDEX parameter (see Fig. 11.7) was the one that the TASK tool can access to activate selected robot moves from inside a companion TSK program (INDEX was therefore equivalent to the PAGE NUMBER when using V.1).

The video file “Video 11.1” showed how to use MOTION V.2 for various functions.

11.3 PC Control of Robot Moves

This little exercise was created to illustrate the basics on integrating PC to MINI communications, TASK and MOTION programs all together in one application.

Most folks likely had used the RoboPlus Manager tool only to update firmware or to have a quick check on actuators and sensors attached to the controller in use (as it was originally intended to). But the Manager tool also had a very handy sub-tool called Zig2Serial Management which was originally created to help manage the ZIG-100 (circa 2009). But as it turned out, this was a very general communications tool that can be used on the PC regardless whether ones use ZigBee or BlueTooth (just use the appropriate Windows COM port—see Fig. 11.8).



Fig. 11.8 Zig2Serial Management sub-tool of RoboPlus Manager

This application used the enclosed files “DARWIN-MINI-1.MTNX” and “DARWIN-MINI-RC.TSK”. The DARWIN-MINI-RC.TSK programming structure was quite simple (see Fig. 11.9):

- Lines 6–7. Set all actuators to JOINT MODE (i.e., “2”) and TORQUE to be ON (TRUE).
- Lines 10–11. The robot next played Motion Index “1” which was the READY Pose.
- Lines 14–23. Then the robot entered an endless loop where it waited for an input number coming from the PC and saved it in parameter “MotionGroupNo” (lines 16–19). The robot sent this “MotionGroupNo” value back to the PC for confirmation (line 20) and triggered this Motion Group’s moves and waited until that was done (lines 21–22).

From practice, the author had found that there was a very particular order that these files had to be downloaded to the MINI for them to work together properly:

- Download the DARWIN-MINI-RC.TSK file first, via the TASK tool and the appropriate COM port. Close the TASK window to release this COM port.
- Download the DARWIN-MINI-1.MTNX file next, via the MOTION V.2 tool and the same COM port. Close the MOTION window and make sure to turn the POWER OFF the MINI.
- Turn power back on the MINI so that the TASK program get executed before starting the Zig2Serial sub-tool from inside the Manager tool. If the reader used BlueTooth, it might take 10–15 s for the Zig2Serial sub-window to come up (see Fig. 11.8)—ZigBee would connect much quicker than BT.

```

3  START PROGRAM
4  {
5      // Set all ID to joint mode
6      ID[254]: ADDR[11(b)] = 2
7      ID[All]: Torque ON/OFF = TRUE
8
9      // Get into Init Pose
10     Motion Index Number = 1
11     WAIT WHILE ( Motion Status == TRUE )
12
13     // Endless loop for RC input from user
14     ENDLESS LOOP
15     {
16         MotionGroupNo = 0
17         WAIT WHILE ( Remocon Data Received == FALSE )
18
19         MotionGroupNo = Remocon RXD
20         Remocon TXD = MotionGroupNo
21         Motion Index Number = MotionGroupNo
22         WAIT WHILE ( Motion Status == TRUE )
23     }
24 }

```

Fig. 11.9 Program DARWIN-MINI-RC.TSK

4. The user could now enter a number into the SEND field of the Zig2Serial sub-window and clicked away on the SEND button. The same number should appear under the SENT DATA list and then also in the RECEIVED DATA list (this indicated that 2-way communications had been established between PC and MINI). This “number” of course had to correspond to a valid INDEX number of the MOTION GROUP that had been downloaded (see Fig. 11.7).

The video file “Video 11.2” illustrated such a session as described above.

11.4 Synchronizing LEDs to Motion

The XL-320 actuators on the MINI were equipped with programmable LEDs. In this next exercise, these LEDs would be programmed to turn on only for those actuators involved in a chosen robot’s move to emphasize this particular move to the audience.

This application would use the “DM-LED-Synch.tsk” files and the Motion Group List named “LED Sync 1” in the “DARWIN-MINI-1.MTNX” file (see Fig. 11.10).

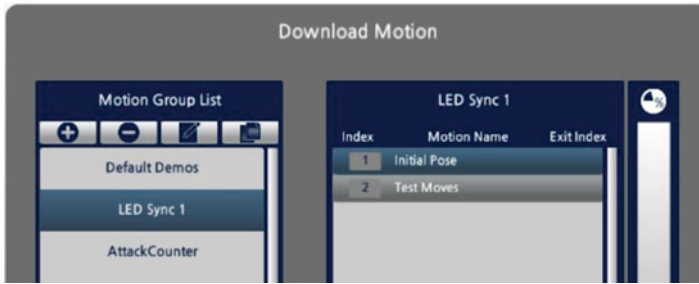


Fig. 11.10 Motion Group List “LED Sync 1”



Fig. 11.11 LEDs ON/OFF timings for Right and Left Arms

This Motion Group List had two Motion Groups labeled “Initial Pose” (Index=1) and “Test Moves” (Index=2).

Figure 11.11 displayed the timing of the Key Frames used in the “Test Moves” Motion Group, and also the ON/OFF timings of the LEDs of the right arm (IDs=1, 3, 5) and of the left arm (IDs=2, 4, 6):

1. Time=[0, 390] ms—The robot turned its head to the right and all LEDs OFF.
2. Time=[390, 796] ms—The robot raised its right arm, thus right LEDs ON.
3. Time=[796, 1195] ms—The robot raised its left arm, thus right LEDs OFF and left LEDs ON.
4. Time>1195 ms—The robot brought both arms down, thus all LEDs OFF.

These four (actually only the first three) time periods were monitored using the Hi-Resolution Timer of the OpenCM-9.04-C to trigger the needed ON/OFF actions for the LEDs involved. In the “DM-LED-Synch.tsk” file, the control logic implemented was quite simple:

1. Play the Motion Group “2” (line 33), and start the Hi-Res Timer for 390 ms and essentially do nothing during this time period as the LEDs were set to OFF at the beginning of this program already (lines 35–36).
2. Lines 39–46—Start the Hi-Res Timer for 406 ms (line 39) and during this time period, turn ON the LEDs of the right arm (IDs=1, 3, 5). When the Hi-Res Timer “timed out”, turn OFF all LEDs (line 46).
3. Lines 49–56—Start the Hi-Res Timer for 399 ms and turn ON the left arm LEDs (IDs=6, 4, 2) and then turn them OFF at time-out (line 56).

The video file “Video 11.3” illustrated the operation of this program using the Zig2Serial Management tool to trigger user-wanted events.



Fig. 11.12 Motion Group List “AttackCounter” with seven Motion Groups

11.5 Fight Choreography for two MINIs via ZigBee

This last application illustrated a possible choreography framework for coordinating the interactions between two MINIs performing some Karate moves. One MINI served as the Lead Fighter while the other acted as the Counter Fighter.

This application used the Motion Group List named “AttackCounter” found in the “DARWIN-MINI-1-ZB.mtnx” file (see Fig. 11.12).

This Motion Group List had seven Motion Groups:

- Index 1 corresponded to the Ready Pose.
- Index 2 corresponded to Attack1 moves while Index 3 corresponded Counter1 moves.
- Indices 4 and 5 corresponded to Attack2 and Counter2 moves, while indices 6 and 7 corresponded Attack3 and Counter3 moves.

The TASK files “DM-LeadFighter-ZB.tsk” and “DM-CounterFighter-ZB.tsk” were designed to work with the previous seven Motion Groups. This application was designed for a ZigBee Broadcast environment whereas the operator/judge would issue a “Ready” command (i.e., a “1”) or a “Fight” command (i.e., an “11”) from the PC via the Zig2Serial Management tool. The interesting feature of the Lead-Fighter program was that it used the Random Number utility available on the OpenCM-9.04 series to trigger at random one of its three possible attack moves, while the Counter-Fighter program would trigger the appropriate non-randomized counter moves (of course the reader can modify the Counter-Fighter code to provide randomized counter moves or added more sensor-based counter moves).

The main logic in the “DM-LeadFighter-ZB.tsk” program was implemented via an endless loop that would “listen” for a ZigBee packet and processed it to figure out whether a “1” or an “11” was actually received:

- If a “1” was received, parameter “MotionGroupNo” was set to “1”.
- If an “11” was received, the controller would throw the dice once and got a value for “RandomNo” between 0 and 255 (line 27). Next, a UNIFORM statistical

```

21      IF ( DataIn == 1 )
22      {
23          MotionGroupNo = DataIn
24      }
25      ELSE IF ( DataIn == 11 )
26      {
27          RandomNo = 2 Random Number
28          IF ( RandomNo <= 85 )
29          {
30              MotionGroupNo = 2
31          }
32          ELSE IF ( RandomNo > 85 && RandomNo <= 170 )
33          {
34              MotionGroupNo = 4
35          }
36          ELSE IF ( RandomNo > 170 && RandomNo <= 255 )
37          {
38              MotionGroupNo = 6
39          }
40          ELSE
41          {
42              MotionGroupNo = 1
43          }

```

Fig. 11.13 Logic for the LeadFighter program to decide on a randomized Attack move to perform

distribution was assumed, thus there would be a 33 % chance for the parameter “MotionGroupNo” to get its final value of 2, 4 or 6 (lines 28–43 and see Fig. 11.13).

- (c) The next step (line 45) was crucial because as we were using a broadcast environment, therefore the PC and the Counter-Fighter would receive all communication packets. Thus we had to “special-code” the information meant for the Counter-Fighter, by shifting it left by 4 bits (i.e., multiply it with 16) and saved this special information as parameter “MotionNoCounterFighter”.
- (d) Then the “LeadFighter” controller was instructed to broadcast the parameter “MotionNoCounterFighter” (line 47) which was really meant for the CounterFighter, and lastly triggered its own Attack move as reflected in the actual value of “MotionGroupNo” (line 50 and see Fig. 11.14).

The main logic in the “DM-CounterFighter-ZB.tsk” program was also implemented via an endless loop that would “listen” for a ZigBee packet and processed it (i.e., divide it by 16—statement 20) to figure out whether “DataIn” was “0” (i.e., coming from the PC) or a “2”, “4” or “6” (i.e., coming from the Lead-Fighter).


```

45      MotionNoCounterFighter = MotionGroupNo * 16
46      // Send to Counter Fighter
47      Remocon TXD = MotionNoCounterFighter
48
49      // Start Fight
50      Motion Index Number = MotionGroupNo
51      WAIT WHILE ( Motion Status == TRUE )
52
53      // Back to Ready Pose
54      Motion Index Number = 1
55      WAIT WHILE ( Motion Status == TRUE )
56  }

```

Fig. 11.14 Logic for the LeadFighter program to send information to CounterFighter

Next this program used that information to set parameter “MotionGroupNo” with the correct Counter Motion Group’s Index value and finally triggered the appropriate Motion Group (see Fig. 11.15).





The video file “Video 11.4” showed the execution of the above programs with two MINIs and within a broadcast ZigBee environment as described above.

11.6 Fight Choreography for two MINIs via BlueTooth

The author also had designed an alternate procedure using R+Motion V.2 under a BlueTooth environment to perform this choreography application.

First, ROBOTIS wrote the R+Motion V.2 in such a way that the Windows PC user can run multiple instances of this tool on their computer (this was not possible with the RoboPlus Motion V.1 and Manager tools). Thus the author’s approach was to spawn out two instances of the R+Motion V.2 application: one instance would control the Lead-Fighter via a given BT outgoing COM port, while the second instance would control the Counter-Fighter via a separate BT outgoing COM port. As the commands to each fighter now came from a single PC, it was more a matter of how fast the user could switch from one window application to the next and click on the appropriate Motion Unit to activate the Attack and Counter moves onto the respective robots (see file “DARWIN-MINI-1-BT.mtnx”). The video file “Video 11.5” illustrated such an episode where the reader could see that “human” manual synchronization of robot moves did not work very well, but that the ROBOTIS-MINI system had lots of potentials.

```

20      DataIn = DataIn / 16
21
22      IF ( DataIn == 0 )
23      {
24          MotionGroupNo = 1
25      }
26      ELSE
27      {
28          IF ( DataIn == 2 )
29          {
30              MotionGroupNo = 3
31          }
32          ELSE IF ( DataIn == 4 )
33          {
34              MotionGroupNo = 5
35          }
36          ELSE IF ( DataIn == 6 )
37          {
38              MotionGroupNo = 7
39          }
40
41          // Start Counter Fight
42           Motion Index Number = MotionGroupNo
43          WAIT WHILE (  Motion Status == TRUE )
44
45          // Back to Ready Pose
46           Motion Index Number = 1
47          WAIT WHILE (  Motion Status == TRUE )

```

Fig. 11.15 Logic for the Counter-Fighter program to decide on an appropriate counter move to perform

Thus it remains a challenge for all users and ROBOTIS to come up with a ZigBee or BlueTooth environment that could be used for multiple MINIs. The author is looking forward to the release of the BT-410 Master-Slave series in the Summer of 2015.

11.7 Review Questions for Chap. 11

1. What are the wireless communication options available with the ROBOTIS-MINI system?
2. What is the highest baud rate achievable with ROBOTIS ZigBee devices?
3. What is the highest baud rate achievable with ROBOTIS BlueTooth devices?
4. List advantages of ZigBee over BlueTooth.
5. List advantages of BlueTooth over ZigBee.
6. List advantages of the MTNX file format over the MTN file format.
7. List the data components found in an MTN file.
8. Describe the data architecture used in an MTN file.
9. List the data components found in an MTNX file.
10. Describe the data architecture used in an MTNX file.
11. Describe how the Zig2Serial Management sub-tool can be used to execute motion pages or groups stored on the ROBOTIS-MINI.
12. List the TASK commands controlling the LEDs found on the XL-320 actuator.

11.8 Review Exercises for Chap. 11

1. Create a custom MTNX file to allow the ROBOTIS-MINI go up and down a set of stairs (see video file “Video 11.6”) (Fig. 11.16).
2. Create a custom MTNX file to allow the ROBOTIS-MINI to dance and synchronize to your favorite music, see example video at <https://www.youtube.com/watch?v=4VsNyzABXsQ>.
3. Combine the example MOTION UNITS into MOTION GROUPS of your own, similarly to the approach used to choreograph the MINI fighters.
4. Integrate sensors such as NIR and DMS sensors to help the MINI avoid obstacles. Later in 2015, ultrasonic and proximity sensors should also be available for the MINI.
5. Practice programming the OpenCM-9.04-C using the ROBOTIS IDE and practice recovering the firmware to get back to programming with TASK and MOTION again.

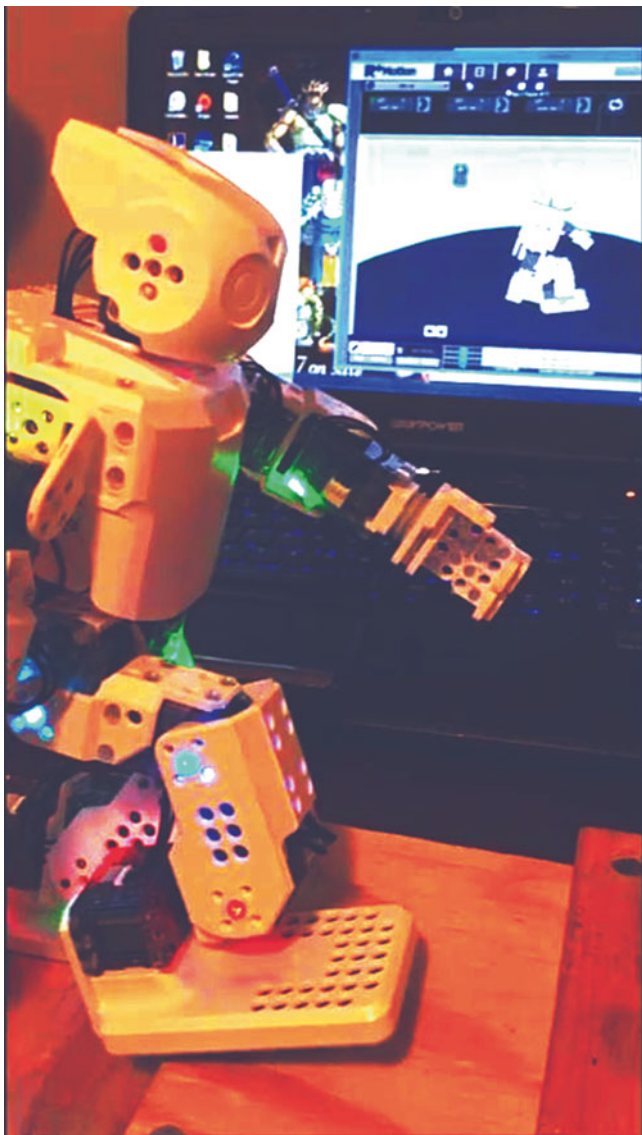


Fig. 11.16 ROBOTIS-MINI going up and down stair steps

References

- Billard A et al (2008) Robot programming by demonstration. In: Siciliano B, Khatib O (eds) Springer handbook of robotics. Springer, Heidelberg, pp 1371–1394
- Calinon S (2009) Robot programming by demonstration: a probabilistic approach. EPFL Press, Lausanne