Programming Guide
For ROBOTIS MINI
(Excerpts)

By Chi N. Thai

CNT Robotics LLC, Buford

2020

Chapter 1: Systemic View of ROBOTIS® MINI™

ROBOTIS releases the MINI kit in 2014 (http://www.robotis.us/robotis-mini/) along with the mobile App called MINI for iOS and Android/Chromebook OS which can be used at a minimum level for entertainment purposes (https://play.google.com/store/apps/details?id=com.robotis.darwinmini, https://itunes.apple.com/us/app/robotis-mini/id948481762). But this kit is much more capable, as shown later in this book, when other software tools such as ROBOTIS R+TASK™/R+MOTION™ or EDBOT® are leveraged for the purpose of teaching robotics fundamentals to students at all ages.

The goal of Chapter 1 is to provide the reader with a systemic view of the MINI kit illustrating how its varied hardware components integrate with a host of software tools at a high conceptual level while leaving the detailed operational details for later chapters.

1.1 Components of MINI kit

In this Section 1.1, the author describes the hardware components that come with the MINI kit when purchased, along with additional hardware components that can be used to enhance the use of this robotics kit.

1.1.1 Hardware Controller: OpenCM9.04-C

The most important component of any robotics kit is of course its "brain", i.e. Hardware Controller, which is the OpenCM9.04-C. The actual microcontroller chip used for the OpenCM9.04-C is an STM32F103CB with 128 KB of Flash Memory – see backside of PCB's picture on the left in Fig. 1.1 (beneath the QC sticker). The partition of this 128 KB memory space into various firmware components will be described in Section 1.3.1.

Fig 1.1 also shows the locations of the components that the user will mostly interact with when using this Hardware Controller:

- The ON-OFF power switch
- Two connectors for the 2-pin cables coming from the two Li-Ion battery cradles LBB-040 or LBB-041 (see Section 1.1.3).
- Four 3-pin Dynamixel™ ports where the 16 servo motors XL-320s will be connected to, in a daisy chain fashion (more details about the term "Dynamixel" are provided in Section 1.1.2). The STM32F103CB microcontroller communicates to the 16 XL-320s via Serial Communications Channel 1 (USART Ch.1) at a high speed of 1 Mbps, as lots of data throughput are needed to control 16 servo motors at the same time.
- Four General-Purpose-Input-Output (GPIO) connectors for various sensors and the LED module (see Section 1.1.6). GPIO ports are connected directly to the I/O ports of the STM32F103CB microcontroller.
- One "external" Serial Communications port where the BT-210 Bluetooth® receiver (and other 3-pin compatible serial communications devices – see Section 1.1.4) can be connected. The STM32F103CB microcontroller communicates to these devices via Serial Communications Channel 2 (USART Ch.2) at a slow speed of 57.6 Kbps, as less data throughput is required for these devices.
- One User Button to be used when recovering firmware for the OpenCM9.04-C and/or the servo motors XL-320 (see Chapter 3).

- One Micro USB-2 port when recovering firmware on the OpenCM9.04-C and its associated XL-320s (see Section 1.3.7 and Chapter 3) or when doing advanced C/C++ programming from a PC using the OpenCM or Arduino IDEs (see Section 1.3.9).

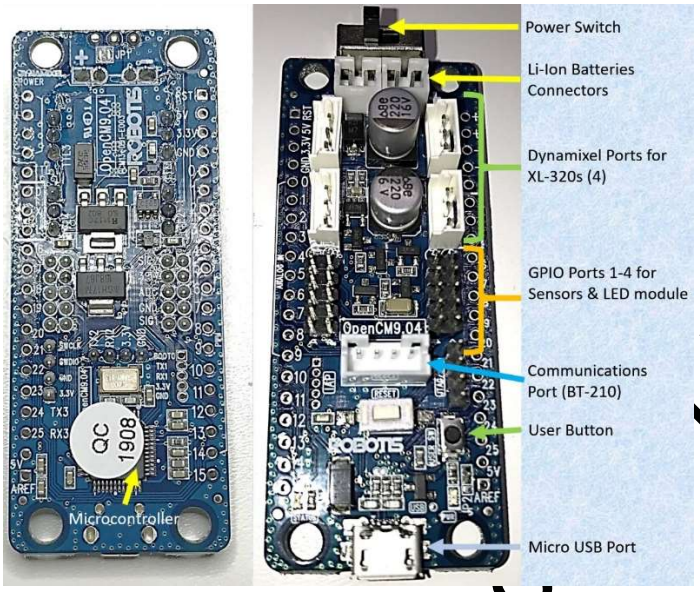

Fig. 1.1 Hardware Controller OpenCM9.04-C.

The reader is referred to the following ROBOTIS web page for more comprehensive details (http://emanual.robotis.com/docs/en/parts/controller/opencm904/) on many other aspects for the proper usage of the OpenCM9.04-C which will be discussed when appropriate in later sections of this book.

ROBOTIS CS also has a YouTube playlist of tutorials for the OpenCM9.04 at https://www.youtube.com/playlist?list=PLEf1s0tzV8nRgF2Cu9r91bU3bHuZU8Fxu

1.1.2 Servo Motor: XL-320 (Dynamixel Concept)

The MINI kit comes with 16 "real" XL-320s which are "smart" servo motors allowing it to perform human-like movements of its arms, legs and hip joints, and 1 "dummy" XL-320D serving as a component of its "unmovable" head assembly. The author would recommend readers to acquire a 17th XL-320 so that this head assembly can be motorized and controlled (see Section 2.2).

ROBOTIS trademarked most of their servo motors (including the XL-320) as "Dynamixel" or "Smart Servo" because each XL-320 has a dedicated microcontroller STM8S105K4T6C controlling the actual servo motor (see Fig. 1.2), and allowing each XL-320 to communicate with the Hardware Controller OpenCM9.04-C within a local wired (3-pin) communications network shared with other XL-320s (i.e. USART Ch.1, at a rate of 1 Mbps), all hooked up as a daisy chain (http://emanual.robotis.com/docs/en/dxl/x/xl320/).

Of course in order to distinguish one XL-320 from another, each XL-320 in the MINI kit was pre-assigned a unique ID from 1 to 16 (see Section 2.2 for how to handle the situation when the reader decides to get that 17th XL-320). This arrangement allows the user to issue a simple assignment command such as *"ID[17]: Goal Position = 512"* to make the Head Servo (ID = 17) to go to Position 512 (i.e. commanding the MINI's head to face forward), without the need to program at the actual hardware level using Pulse-Width-Modulation (PWM).

Moreover, Hardware Controllers such as the OpenCM9.04-C are also considered as Dynamixels with a specially assigned **ID = 200** when using ROBOTIS provided software such as R+TASK, R+MOTION and OpenCM IDE, and consequently there can only be **<u>one</u>** Hardware Controller within each local wired Dynamixel network (i.e. for each ROBOTIS robot). In other words, the user should not use **ID = 200** for any of the XL-320s used.

Fig. 1.2 Printed Circuit Board (PCB) for XL-320.

The XL-320 is capable of 2 modes of operations: Continuous-Turn (i.e. as a wheel spinning CW or CCW) or Position-Control (i.e. as a true servo maintaining a given rotational position), but the Continuous-Turn mode is obviously not appropriate for the MINI kit which is a humanoid robot. However, the author will demonstrate the use of the 17th XL-320 (Head's servo) in both modes for the reader's benefit in Section 6.3.

Let's next look into how does the XL-320 "know its rotational position" at any one time:

- The left picture in Fig. 1.3 shows the inner details of an XL-320 that had been disassembled to show that when the Servo Motor turns, the Gear Train transmits this rotational motion into the Horn (i.e. to move the MINI's head for example) and also into the Position Encoder below it.
- The middle picture in Fig. 1.3 shows the PCB with the Position Encoder's cover removed (far right picture) so that its actual mechanism is exposed. This Position Encoder is essentially a rotational potentiometer using 2 graphite rings. The inner ring is continuous and serves as an analog voltage input signal for the microcontroller STM8S105K4T6C to digitize and send the resulting numerical value as "Present Position" to the OpenCM9.04-C. This "Present Position" value can then be used in the user's control program. The outer ring is "discontinuous" with one end connected to electrical ground and the other connected to V_{DD} = 7.4V, and because of this discontinuity the XL-320 can only provide positional information for only an arc of 300 degrees out of 360 degrees possible for a complete circle.

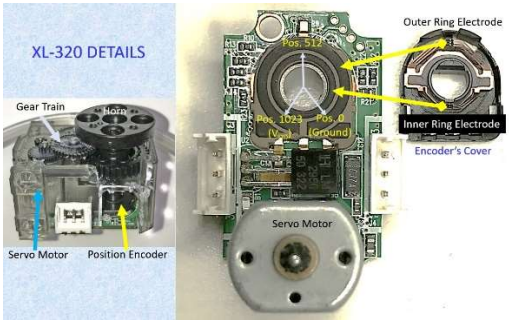

Fig. 1.3 Details of a Disassembled XL-320.

1.1.3 Batteries: LB-040/041

1.1.4 Communication Hardware: BT-210, USB & Others

1.1.5 Frame Parts & Mechanical Assembly

1.1.6 Compatible Actuators & Sensors (Port Concept)

The MINI kit does not include any sensor at all, but the OpenCM9.04-C can handle a variety of sensors and an LED module through four 5-pin GPIO connectors as shown in Fig. 1.1. For a list of compatible devices for each specific connector/port please check this web link (http://emanual.robotis.com/docs/en/parts/controller/opencm904/#robotis-5-pin-port). For example, in this book the author uses 2 NIR sensors IRSS-10 (http://www.robotis.us/ir-sensor-irss-10/) mounted on either side of the MINI's head and a DMS-80 on the top of its head (http://www.robotis.us/distance-measuring-sensor-dms-80/) (see Fig. 1.4). Internally, the IRSS-10s are connected to Port 1 and Port 4, while the DMS-80 is connected to Port 2 and typical programming commands to access them are of this type *"Left_Head_Sensor = PORT[1]:IR Sensor"*, i.e. to read Port 1 for an IR Sensor input and to save this digitized value into Parameter "Left_Head_Sensor".

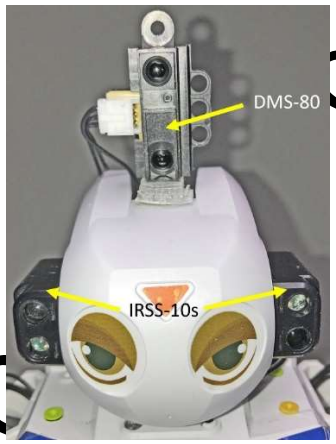

Fig. 1.4 MINI robot Lina with 2 IRSS-10s and 1 DMS-80 mounted on her head.

1.2 MINI Software Tools

There is a plethora of software tools that work with the MINI and they can come from ROBOTIS as free downloads or from other organizations or vendors which can be free or need to be purchased separately (see Section 1.2.2). ROBOTIS tools of course have extensive documentations at their e-manual web site http://emanual.robotis.com/ and users can also register at a public forum http://en.robotis.com/service/forum.php to get official answers to any technical issue that they may have.

1.2.1 ROBOTIS Provided Software

ROBOTIS offers many software tools that work with the MINI for free at their web site http://en.robotis.com/service/downloadpage.php?ca_id=10, notably:

- R+MANAGER V.2 which works only on Windows OS (or macOS with appropriate Windows Virtualization software).

- R+TASK and R+MOTION used to be 2 separate components for Versions 1 and 2 (Windows, Linux and macOS). In Summer 2019, ROBOTIS decides to upgrade and combine these two packages together and calls it R+TASK V.3 (see Fig. 1.5). R+TASK V.3 works with Windows, Android/Chromebook and macOS. **This book uses R+TASK V.3** and the Windows version can be directly downloaded from this link (http://www.robotis.com/service/download.php?no=1774). Mobile versions of R+TASK V.3 can be downloaded from these links (https://play.google.com/store/apps/developer?id=ROBOTIS or https://apps.apple.com/us/developer/robotis-co-ltd/id948481761).
- R+DESIGN as previously mentioned in Section 1.1.5 works on Windows, iOS and Android/Chromebook.
- DYNAMIXEL WIZARD V.2 which works with Windows, Linux and macOS (http://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_wizard2/).
- For Mobile Apps (iOS and Android/Chromebook), the reader is already aware of the ROBOTIS MINI App, but the reader probably does not know that the R+m.PLAY700 App (iOS and Android/Chromebook), originally designed for the PLAY700 kit (i.e. Hardware Controller CM-50), does work fine with the OpenCM9.04-C using the TASK's SMART DEVICE features (see Chapter 7). The reader can download R+m.PLAY700 to his or hers compatible mobile device at these web links (https://apps.apple.com/us/app/play700/id1156037721, or https://play.google.com/store/apps/details?id=com.robotis.play700).
- For more advanced users using C/C++, they can use the package OpenCM IDE (http://emanual.robotis.com/docs/en/software/opencm_ide/getting_started/) which can be programmed to work with the R+m.PLAY700 App. OpenCM IDE works on Windows, Linux and macOS, unfortunately ROBOTIS is no longer supporting this package and is promoting the use of standard ARDUINO interface with their new tools called Dynamixel SDK and Dynamixel Workbench (http://emanual.robotis.com/docs/en/software/arduino_ide/) which work on Windows, Linux and MacOS.

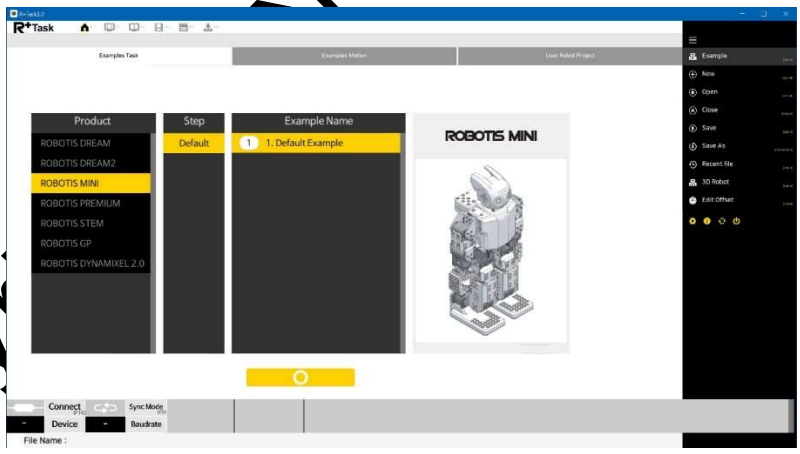

Fig. 1.5 Main Menu of the new R+TASK V.3.

1.2.2 ROBOTIS Partner Software

Currently, the author is aware of only 1 commercial software tool that works with the MINI and it is called EDBOT from Robots in Schools Ltd. (http://ed.bot/edbotsoftware). EDBOT runs on Windows, Mac, Linux (including Raspberry Pi) and Android/Chromebook. EDBOT allows the use of several development

tools such as SCRATCH 2/3, Python, JavaScript and C# (http://support.ed.bot/) to work with ROBOTIS firmware and Motion files. A unique EDBOT feature is that it will allow the user to control several MINIs in the same user program in the languages listed above (https://www.youtube.com/watch?v=yUpx42eNR2I). Chapter 8 of this book will provide the reader with a firm foundation in using EDBOT V.5 with SCRATCH and Python.

The Synthiam organization supports the MINI with their free EZ-Builder environment (compatible with ARDUINO IDE tools from ROBOTIS) but it requires a different firmware than the standard one from ROBOTIS (https://synthiam.com/GettingStarted/Build-Robots/Robotis-OpenCM-9-04-17530). This is a good project for an advanced reader who wants to go beyond the standard MINI, with video camera control and speech recognition for example (https://synthiam.com/GettingStarted/Build-Robots/Robotis-Darwin-Mini-17529).

The European Poppy Project uses the XL-320s but with their own controller (Raspberry Pi), frame parts and free development software environments (https://forum.poppy-project.org/t/getting-started-with-poppy-project/362). This is also a good project for an advanced reader who wants to go beyond the standard MINI.

1.3 MINI Operating Configurations

Section 1.2 shows that there is quite a variety of software tools that can work with the MINI, and specific configurations of the firmware components on the OpenCM9.04-C will be needed for these software tools to work properly. **In this Section 1.3, the author will limit the discussion to the operational characteristics of ROBOTIS provided software tools and EDBOT V.5.**

The most important point of reference to understand the various MINI operating configurations is the Memory Map of its 128 KB of Flash Memory, so that is where we are going to start next.

1.3.1 Memory Map of OpenCM9.04-C

The Hardware Controller (HC) OpenCM9.04-C has 128 KB of Flash Memory partitioned out as shown in Fig. 1.6:

- The Boot Loader and EEPROM memory sections cannot be altered by the user.
- When the MANAGER tool is used to recover or update the OpenCM9.04-C's firmware, the memory sections from Address 0800-3000 and up to 0801-F800 are overwritten with new data (see Section 1.3.7).

Firmware Component	OpenCM9.04-C	
	Start Address	Size in KiloBytes
Boot Loader	0800 0000	12
ROBOTIS Firmware	0800 3000	42
ROBOTIS TASK	0800 D800	8
ROBOTIS MOTION	0800 F800	64
EEPROM	0801 F800	2

Fig. 1.6 Memory Map for the 128 KB Flash Memory of OpenCM9.04-C.

1.3.2 Using MOTION 3 Component

The MOTION and TASK tools are designed to work together for a humanoid robot such as the MINI, but in the previous 2 versions they were provided as 2 separate applications using the same Serial Communication Port, so it was quite bothersome to switch back and forth to the proper COM Port when the user wants to develop a custom MINI project which usually requires both TASK and MOTION tools to be running. Then in Summer 2019, ROBOTIS releases R+TASK V.3 which combines TASK and MOTION into 1 application (see Figs. 1.8 and 1.11) and resolves that long-standing annoyance (**if the user does not use the "Synch" mode** inside MOTION 3 Component - for more details on the "Synch" mode see Chapter 4).

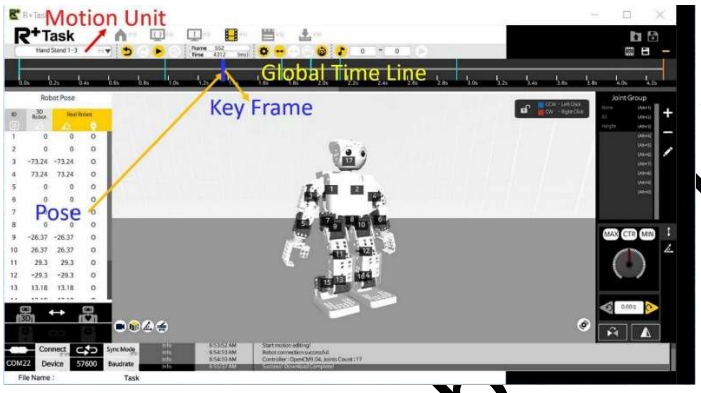

Fig. 1.8 MOTION 3 Component of R+TASK V.3.

1.3.3 Using MINI Mobile App

ROBOTIS has published instructional information for the MINI Mobile App at this link http://emanual.robotis.com/docs/en/software/mobile_app/mini_app/.

The user can use this App to do several basic tasks:

- Do an Assembly Check of the MINI and adjust each servo's OFFSET value.
- Associate a button on this App Interface to a specific MOTION LIST via its DEFAULT MOTION INDEX NUMBER. Thus, the user needs to make sure to use the MOTION tool to download the standard example MOTION GROUP to the MINI before using this App (especially if the user somehow had to recover the firmware on the OpenCM9.04-C, as this action would erase all existing MOTION data and TASK code).
- Associate a user gesture to a specific MOTION LIST.
- To use Voice Recognition to activate a specific MOTION LIST.
- To use Messenger's notifications to activate specific MOTION LISTs.
- This App also offers a Virtual Controller interface for the user to access the Soccer or Fighter modes.

1.3.4 Using TASK 3 Component

As previously mentioned, the new R+TASK V.3 application combines into one interface the two tools TASK and MOTION that used to be separate software packages. In this sub-section, the TASK 3 Component is described in general concepts and features (see Fig. 1.11) while Chapter 6 will take the reader into more detailed usage procedures.

TASK V.3 kept the functionalities of TASK V.2 which are documented at this link (http://emanual.robotis.com/docs/en/software/rplus2/task/) and added a new command for the OpenCM9.04-C controller:

- POLYNOMIAL, i.e. " **? = ? + (? - ?) * ?** "

Please note that TASK is the "true" programming tool to use for controlling the algorithmic behavior of the robot, while MOTION is more of a generator of data as MOTION UNITs and MOTION LISTs:

- The TASK 3 Component provides all programming flow control structures such as Sequence, Condition, Loop and Function, but no explicit arrays are supported. It provides one CALLBACK function which is activated by the hardware timer every 7.81 ms (which interestingly is the same value for the smallest TIME STEP allowed between consecutive MOTION KEY FRAMES – see Section 1.3.2).

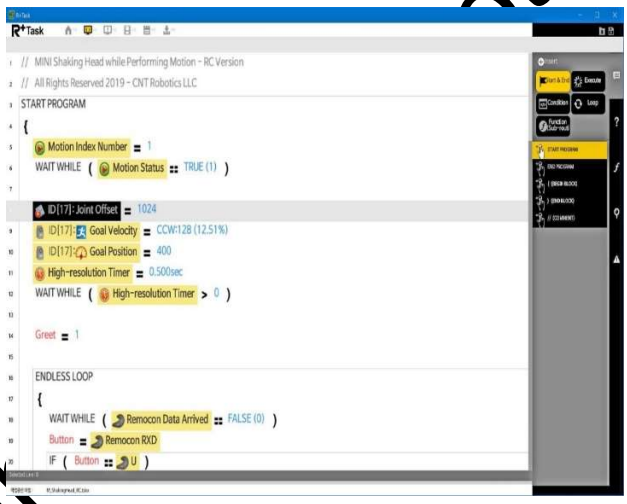

Fig. 1.11 TASK Component of R+TASK V.3.

1.3.5 Using PLAY700 Mobile App (with TASK)

The R+mPLAY700 Mobile App was released in 2016 and it was originally designed for the CM-50 which is the Hardware Controller for the PLAY700 kit (http://www.robotis.us/robotis-play-700-ollobot/, http://emanual.robotis.com/docs/en/edu/play/play-700/), but the OpenCM9.04-C can use this Mobile App as well via the SMART DEVICE commands as part of a TASK program (see Fig. 1.13 and YouTube video https://www.youtube.com/watch?v=Z0JeShxiolc). Chapter 7 will take the reader into the details of using these SMART DEVICE functions.

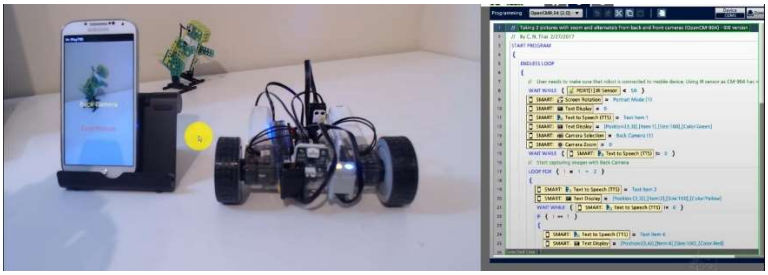

Fig. 1.13 Example of an OpenCM9.04-C carbot using TASK V.2 and R+m.PLAY700

1.3.6 Using EDBOT 5

EDBOT (http://ed.bot/edbot) is a software tool designed to use ROBOTIS MOTION data and to issue direct commands to the OpenCM9.04-C Firmware with non-ROBOTIS software such as SCRATCH, Python, JavaScript or C# (i.e. **EDBOT puts the OpenCM9.04-C into "Manage" Mode**). It can be purchased at https://shop.ed.bot/collections/products separately if the user already possessed a MINI kit (https://shop.ed.bot/collections/products/products/edbot-software), or together with a completely assembled MINI robot equipped with one NIR sensor (https://shop.ed.bot/collections/products/products/edbot).

EDBOT is currently at Version 5 and it can be freely downloaded from http://support.ed.bot/edbot-software-download.html but it won't work properly until the user obtains an install key from the parent company Robots in Schools Ltd. EDBOT works on Windows, Linux, macOS and Chromebook. It is a "server" tool so it can be used to control several MINI robots connected within the same Bluetooth network (see Fig. 1.14 and YouTube video at https://www.youtube.com/watch?v=yUpx42eNR2I). Chapter 8 will show the reader how to use the EDBOT tool with SCRATCH and PYTHON.

Fig. 1.14 Example of a SCRATCH application with EDBOT Tool.

1.3.7 Using MANAGER 2 (Windows OS)

1.3.8 Using DYNAMIXEL WIZARD 2

1.3.9 Using OpenCM IDE & Arduino IDE

1.3.10 Using OLLOBOT SDK

1.4 Dynamixel System Design Paradigm

By now, the reader is probably overwhelmed by the various options to operate the MINI, thus the goal of this section is to provide an integrative view of all ROBOTIS Educational Robotics kits - a sort of a zoom out to see the "forest" so that the reader won't feel lost among the "trees"!

To be clear, the following materials are the author's personal view and understanding of ROBOTIS systems from his many years of using them. Officially, ROBOTIS so far had not published any details about their robotics system design approach, although they did patent the concept of "Dynamixel".

The author believes that the MINI kit shares into a common ROBOTIS Dynamixel System Design Paradigm which considers a typical ROBOTIS robot to be a Computer Network with four major components:

1) A Hardware Controller, e.g. OpenCM9.04-C for the MINI, which is the "Main Brain" for the robot (Dynamixel ID = 200). It contains a firmware to allow low-level functionalities and additionally stores the user-programmed instructions for the robot to execute at run time.

2) The Sensors component which helps the Hardware Controller get information about its environment, from a simple sensor such as the Color Sensor RCS-10 (connected to local I/O ports of Hardware Controller) to a more complex sensor such as the IR Sensor Array (from Bioloid STEM kit) which requires its own embedded controller with its unique Dynamixel ID.

3) The Actuators component which allows the Hardware Controller to perform the appropriate robot actions upon the real world as programmed by the user, from a simple geared motor such as the GM-10A to the more sophisticated dual servo module such as the 2XL430-W250-T with its own embedded controller and unique Dynamixel IDs.

4) A Remote Device (Dynamixel ID = 100) which can be a mobile device or even a desktop PC. The Remote Device has a more flexible role depending on the specific device used and on its current role in this network of robotic components. For example, if a smartphone (running the MINI App) is used with the OpenCM9.04-C, it can act as a BlueTooth Remote Controller (i.e. a "glorified" Sensor) telling the Hardware Controller about which U-D-L-R Touch Area on the phone screen had been pressed by the user. However, if the same smartphone (running the R+ PLAY700 App) is used with the MINI, it can additionally act as an Actuator displaying appropriate graphics or videos as commanded by the OpenCM9.04-C according to its TASK programmed logic (see Chapter 7). When a desktop PC is used as a Remote Device, the interactions become more complex. For example, if the TASK's Virtual Remote Controller is used (see Fig. 1.16), then the PC acts as a glorified Sensor informing the Hardware Controller about which "U-D-L-R-1-2-3-4-5-6" buttons had been pushed by the user. However, if the SCRATCH-PYTHON/EDBOT tool chain is instead used on the PC (see Chapter 8), then the PC takes on the role of a Primary Hardware Controller as SCRATCH-PYTHON codes run on the PC, and not on the OpenCM9.04-C which now has a secondary role as it is just letting SCRATCH-PYTHON commands flow from the PC (via EDBOT) to pass through to its attached Actuators and Sensors (via the OpenCM9.04-C's Firmware).

Chapter 2: Kit Assembly Tips

The MINI kit comes with a Quick Start manual to guide the user in the assembly of the MINI robot. The author recommends the user to also use the R+DESIGN software for certain steps that may be unclear in this Quick Start manual as it can only show **one view angle** of the 3D sub-assemblies, while this view angle can be controlled by the user inside R+DESIGN (see Fig. 2.1 and Video 2.1). However only the Quick Start manual would show the proper routing of the 3-pin cables connecting all the XL-320s into a daisy chain.

The goal of Chapter 2 is to offer some practical information to make the robot construction process easier for the first-time builder. The reader is recommended to read Section 2.1 before starting on building the MINI robot for the first time.

2.1 Preparations Before Assembly

Depending on the user's needs such as planning to use the Micro-USB port or to motorize the head piece and/or use extra ROBOTIS sensors, certain steps have to be performed in advance of the construction phase of the MINI robot as shown in the Quick Start manual.

2.1.1 Creating USB Access Slot

If in addition to the regular MANAGER/TASK/MOTION tools, an advanced user plans to use the OpenCM IDE, then this user would need to access the micro USB port on the OpenCM9.04-C at all times and this will require the user to create an access slot on the MINI torso frame part DMF-B01 (see Fig. 2.2). This slot can be created using a drill bit or any other rotating carving tool.

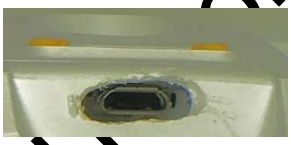

Fig. 2.2 Micro USB slot carved into MINI frame part DMF-B01.

2.1.2 Firmware Update and Recovery

2.1.3 Dynamixel Check

2.1.4 Optional Sensors & Cables

Fig. 2.5 Attaching IRSS-10 and DMS-80 Sensors to MINI Head Part.

2.2 Assembly of Head Servo

The MINI kit comes with a "dummy" XL-320 for its Head subassembly, but for this book the author needs it to be motorized so it can be swung right and left, and along with the NIR Sensors on both sides of its head (Section 2.1.4), an "object scanning" feature can then be developed (see Chapter 6).

The author had created a video to help interested readers with the addition of this Head Servo to the standard MINI robot at https://www.youtube.com/watch?v=rVBDah3DjDQ.

2.3 Mobile Device Attachment

To use the R+m.PLAY700 with the MINI (via TASK programming – see Chapter 7), ones would need to use a smart phone (either Android or iPhone). The author used some Dual-Lock tape to attach an older Samsung S2 to the robot (see Fig. 2.6) and the reader can see that the extra weight made the robot lean further forward. This effect would need a readjustment of the various **offset** values of the **servos serving as ankle, knee and hip points of the MINI before the default MOTION PAGEs, as provided by the ROBOTIS MOTION tool, can be used (see Section 4.4 and Chapter 7 for more details)**, especially the ones used for walking. Thus, the user is recommended to use the smallest and lightest smartphone possible.

Fig. 2.6 Attaching a mobile device to the MINI robot will make it lean more forward.

2.4 Post-Assembly Troubleshooting

Chapter 3: Using MANAGER (Windows OS only)

The MANAGER V.2 tool (http://emanual.robotis.com/docs/en/software/rplus2/manager/) is designed to allow the user to perform a comprehensive hardware/firmware check on the entire MINI robot. This capability had been useful to the author to troubleshoot actuators and sensors when they did not seem to work properly, and to determine that it was due to actual hardware malfunctions and not to some programming errors that the author may have done inside TASK or other programming IDEs.

The goal of this Chapter is to get the reader familiarized with the main features of the tool MANAGER V.2 and to show how they are related to the complete Control Tables of the OpenCM9.04-C and individual XL-320 actuators as well as other ROBOTIS sensors. The reader is also shown a preview of how to access some features of R+TASK V.3 before using them fully later in Chapters 4 and 6.

3.1 Connection Options between PC and MINI robot

The MINI kit comes with the BT-210 Bluetooth Receiver and a micro USB Cable which can be used to connect to the MINI's Hardware Controller OpenCM9.04-C, however there are other connection options that provide other advantages (and disadvantages) such as the BT-410 Bluetooth Set and LN-101 USB Dongle.

3.1.1 Using BT-210

The BT-210 is provided to the user with the purchased MINI kit and it is based on Bluetooth 2.0 which uses more electrical power from the Hardware Controller that it's connected to. Furthermore, all users of the BT-210 could not use this receiver with ROBOTIS provided software for most of the year 2018 due to a Windows 10 update which was not compatible with the BT-210. Since Spring 2019, ROBOTIS upgrades the BT-210's firmware and this BT-210's problem no longer exists with ROBOTIS controllers such as the MINI, but who knows, this issue may pop up again in the future! Pairing the PC's Bluetooth server to the BT-210 receiver on the bot is done via the normal Bluetooth setup tool from Windows OS.

Fig. 3.1 shows that the BT-210 would show up as a "Bluetooth Serial Port" inside MANAGER which only uses the 57.6 kbps rate although quite a few connection speeds are available for the BT-210 (http://emanual.robotis.com/docs/en/parts/communication/bt-210/#specifications).

The BT-210 is not compatible with iOS devices, so if the reader intends to use iOS devices and the R+m mobile versions of TASK and MOTION or the MINI App, then the reader needs to invest in getting the BT-410 receiver and the BT-410 USB dongle (see Section 3.1.2).

3.1.2 Using BT-410 and BT-410 Dongle

The user may already have the BT-410 (http://emanual.robotis.com/docs/en/parts/communication/bt-410/) because he or she wanted to use the MINI mobile App on iOS devices such as iPhones and iPad. Then in order to use MANAGER on a Windows PC, the user would need to also purchase the BT-410 USB Dongle (http://emanual.robotis.com/docs/en/parts/communication/bt-410-dongle/). The BT-410 Dongle would

3.1.3 Using Micro-USB Cable

A **good** Micro-USB cable, i.e. one capable of doing data transfer reliably in addition of a power charging capability, can allow MANAGER to connect to the OpenCM9.04-C Controller from a PC. However, the Micro-USB cable is incompatible with the BT-210 and BT-410 receivers, so the author uses the micro-USB cable with MANAGER only in the situation when all Bluetooth connections are not possible with the OpenCM9.04-C. Furthermore, using the micro-USB cable requires the practitioner to create an access slot to the torso frame part of the MINI robot (see Section 2.1.1). The OpenCM9.04-C would show up as

"ROBOTIS Virtual COM Port" when the Micro USB connects the PC to the OpenCM-9.04-C, and only the 57.6 Kbps rate is allowed with the micro-USB cable via MANAGER (see Fig. 3.3).

3.1.4 Using LN-101

The LN-101 (http://emanual.robotis.com/docs/en/parts/interface/ln-101/) is a USB device and the second wired option that the user can use when the Bluetooth options are not possible. The BT-210/410 Receivers also need to be removed from the OpenCM9.04-C's 4-pin Communication Connector (see Fig. 1.1), because the 4-pin cable of the LN-101 will need to go to that same connector. From the author's personal experiences, he had found that the micro-USB cable connection gets flimsy with use, while the LN-101 keeps good pin connections always.

3.2 Main Features of MANAGER V.2

Three main features are shown for MANAGER in Fig. 3.5, although only "Update & Test" and "Firmware Recovery" are currently operational. The "Self-Checklist" option just takes the user to a temporary web site.

3.2.1 Using "Update & Test"

3.2.2 Using "Firmware Recovery"

3.3 Using MANAGER on completed MINI

In this situation, it is best to lay the MINI flat on its back or its front side before turning it on and connect MANAGER to it to prevent any unforeseen servo malfunction from making the MINI collapse and damage itself. Once MANAGER connects successfully to the OpenCM9.04-C, the user should see a screen such as the one shown in Fig. 3.7 and the user needs to remember to scroll down to see all servos listed. If some servos are missing, this may mean that some cable connections may be loose or that some servos have become non-operational, and sometimes a firmware recovery would solve this later problem but remember the "ID = 1" issue mentioned in Section 3.2.2. Once again, the user needs to remember that the full range of servo positions [0 – 1023] are not available due the mechanical constraints of the MINI's framework, i.e. the user should try to vary the servos' positions only a little bit at a time while watching for possible jam situations.

Chapter 4: Using MOTION Tool

As previously mentioned in Chapter 1, the current V.3 of the MOTION Tool is combined with the TASK component into a single application called R+TASK V.3, available in Windows, Android and Mac OSes (http://www.robotis.com/service/downloadpage.php?ca_id=10). This integration allows a more efficient way to create, test and program MOTION data sets. The technical manual for R+TASK V.3 is available at (http://emanual.robotis.com/docs/en/software/rplustask3/). This e-manual contains the step-by-step instructions for all basic features of the TASK and MOTION Components for R+TASK V.3, thus this book will not try to re-write these How-To's but only to refer to or expand on specific ones as needed.

The goal of this Chapter is to get the reader familiarized with the concept of "MOTION" as implemented by ROBOTIS and get in some practices in creating KEY-FRAME, MOTION-UNIT, MOTION-LIST and MOTION-GROUP.

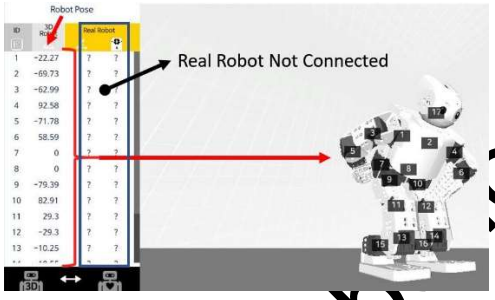

Fig. 4.1 A predefined POSE/KEY-FRAME from the "Greet 2" MOTION-UNIT.

First, the reader is recommended to view Video 4.2 for a quick tour of the Motion Editing features of the MOTION V.3 tool before reading on the rest of this chapter.

The ROBOTIS MOTION data structure is quite complex and requires several layers of understanding to be proficient at it. There are two fundamental concepts that need to be understood clearly: a POSE/KEY-FRAME and a TIME-LINE (i.e. timing) for these KEY-FRAMEs:

- Creating head/arm/leg movements for the MINI is very similar to making cartoons which are essentially multiple "sequential" frames of "static images" that have minor changes done to the "object(s) in motion" in each frame. Each of these static images of the MINI is called a POSE which can be defined as a 1-dimensional array of positional values (i.e. angle values from -300 degrees to +300 degrees) of the various servos used on the MINI. This array uses the servos' IDs (1 to 17 in the case of the MINI with head joint) as its array index (see left side of Fig. 4.1).
- When the physical MINI robot performs these predesigned movements in real time, the OpenCM9.04-C needs to have the poses for each of its servos updated every 7.81 ms (a hardware requirement) (*Note: human muscle signals get updated every 4 to 50 ms*). Fortunately, the MOTION tool does not require the user to predefine robot poses every 7.81 ms, so the user needs to only define Key Poses, also referred to as KEY-FRAMEs, on a TIME-LINE every few seconds apart or so (see Fig. 4.3). Then, at run-time, the OpenCM9.04-C firmware will generate as needed INTERPOLATED-FRAMEs between adjacent KEY-FRAMEs for every 7.81 ms time interval. This procedure "guarantees" that going from 1 KEY FRAME to the next will keep up with the time schedule of KEY-FRAMEs as defined by the user on the TIME-LINE.

4.1 Motion Data Structure and Motion Design

For this section, we'll just use the simulated 3D graphics MINI robot and we won't need the real MINI until Section 4.4. Let's first explore various components provided by the MOTION tool to make this Motion Programming work possible using the Default Example Motion File for the MINI, Thus would the reader please run the R+TASK V.3 application and open up the MINI's Example Motion File with Head Joint (see Fig. 4.2). This would take the user to the MOTION-UNIT Window (partially shown in Fig. 4.3) which can also be reached from anywhere inside TASK 3 by pushing F4 on the keyboard.

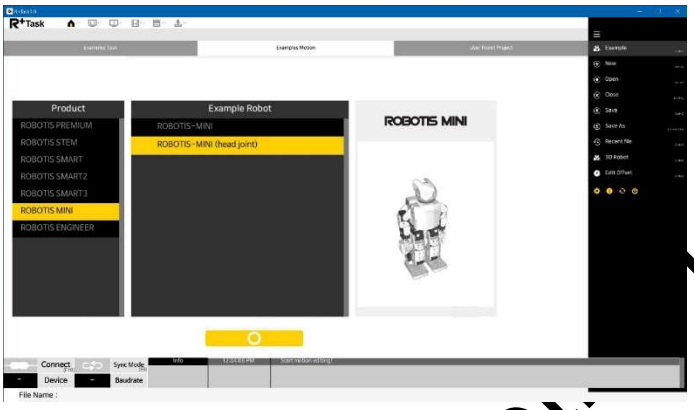

Fig. 4.2 Opening the MINI's Example Motion File with Head Joint.

4.1.1 Key-Frame, Time-Line & Motion-Unit

4.1.2 Basic Motion Design Concepts

4.2 Motion-Units' Flow Control with Motion-List Tool

For this section, we'll also just use the 3D graphics simulated MINI robot. In a way, we can consider the MOTION-UNITs to be the different "MOTION LETTERS" in a user-defined "MOTION ALPHABET", and the MOTION-LIST is the tool to create "MOTION WORDS" from these "MOTION LETTERS". The MOTION-LIST tool can be accessed from anywhere inside TASK 3 by pushing the F5 key (see Fig. 4.14). Officially ROBOTIS named this tool "MOTION", but the author added "LIST" to create "MOTION-LIST" to distinguish it from "MOTION-UNIT" and "MOTION-GROUP". In older ROBOTIS documents (e.g. for TASK V.1) , "MOTION-PAGE" was used to refer to the same data construct as "MOTION/MOTION-LIST".

4.2.1 Basic Use of Motion (List) Tool

The MOTION LIST tool by default opens on the first item which happens to have only 1 MOTION-UNIT i.e "Initial Position", of course other MOTION-LIST items may have a combination of MOTION-UNITS as shown in later paragraphs. The most used menu buttons are shown in Fig. 4.14:

4.3 Motion Lists Organization with Motion Group

The next level of organization is with the MOTION-GROUP tool which can be accessed by pushing F6. The MOTION-GROUP tool can be used to gather user-chosen MOTION-LIST Items into different MOTION-GROUPs, although only one MOTION-GROUP can be downloaded to the MINI at any one time (see Fig. 4.22).

4.4 Servo Offset Calibration

In this section, we will need to use the real MINI robot in synchronized mode with the MOTION Tool which can be achieved with the following procedure described in Fig. 4.25:

- Push F10 to connect MOTION to the MINI with the proper Bluetooth COM Port.
- Next, push F9 to synchronize the 3D graphical robot to the real robot.
- Next, PLAY the MOTION-UNIT named "Calibration" provided by the author in his file "MINI_Custom.mtn3" (it is located towards the bottom of the Motion Units list).
- Then, if the MINI robot had been well constructed, the real robot would match its pose with the 3D graphical robot as shown in Fig. 4.25. This means that for this given MINI robot, the existing OFFSET POSE is properly set, but where can we find this so-called OFFSET POSE?

4.5 Motion Creation Practicum

When the author first saw the Motion Unit "Greet 2" (see Fig. 4.30), it reminded him of a "Crouching Bow" move made by practitioners of one of the Escrima schools in the Phillippines whereas:

1. The arms usage is reversed, i.e. left arm in the back and right arm in front.
2. The right first is also raised to the forehead level.
3. The left leg is shifted to the back to yield a slight crouch of the body with the right knee bent.

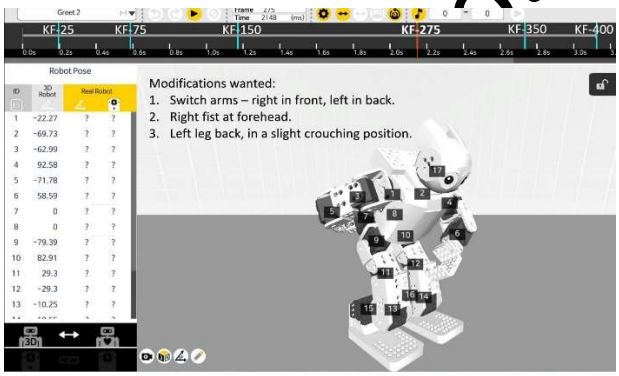

Fig. 4.30 Modifications to be made to Motion Unit "Greet2" to create a "Crouching Bow".

Let's first review how a human would perform this "Crouching Bow":

1. From a feet together standing position, a human shifts his/her weight to the right leg and curl the toes of the left leg to reduce its contact area with the ground and to get it ready to be slid back to the rear.
2. As the left leg slides to the back, the right knee also bends to lower the whole body. At the same time the arms move appropriately (left arm to the back and right arm to the front).
3. The moves timing would have to be such when the body gets to the final crouching position, the right first also reaches the forehead level (i.e. key frame KF-150 in Fig. 4.30).

The author immediately realized that the MINI robot had "rigid" feet, i.e. "no toes curling" was possible, so he had to figure out a different way to reduce the contact area of the MINI left leg in order to slide it backwards. In theory, the rest of the maneuvers should be feasible via the MOTION Tool from what we have seen so far. The reader may also review various editing techniques by consulting with the materials presented at this web link http://emanual.robotis.com/docs/en/software/rplus2/motion/#edit-motion-unit3d-robot before continuing with the rest of this Chapter.

The author divided this project in three parts:

1. Modify "Greet 2" into "Crouching Bow 1" whereas the bowing moves stayed the same except that the arms were switched so that the left arm went to the back while the right arm went to the front (with no changes to the feet servos).
2. Modify "Crouching Bow 1" into "Crouching Bow 2" whereas KEY-FRAMEs KF-150 and KF-275 would now make the MINI robot raise its right first to its forehead (also with no changes to the feet servos).
3. Modify "Crouching Bow 2" into "Crouching Bow 3" by adding the feet maneuvers to get the MINI robot into a slight crouching position by KF-150 and then to stand back up by KF-400. This would be the most complicated phase of this project as the real MINI robot could not be allowed to fall during these leg moves.

4.5.1 From "Greet 2" to "Crouching Bow 1"

4.5.2 From "Crouching Bow 1" to "Crouching Bow 2"

4.5.3 From "Crouching Bow 2" to "Crouching Bow 3"

4.6 Motion Editing vs. Motion Programming

ROBOTIS literature has always been careful to use the words "MOTION EDITING" when referring to the usage of the MOTION tool, and the MOTION FILE is considered as DATA. Only a TASK file (see Chapter 6) is considered as a "real" robot control PROGRAM. But in the author's opinion, the MOTION tool does support the 3 fundamental control structures used in all computer programming languages, albeit in a limited way:

1. SEQUENCE CONTROL – When working with MOTION-UNITs, the user learned how to sequence the robot's poses in a choreography sense instead of a mathematical logic sense as in standard computer programming (see Fig. 4.3). The user also learned how to account for the laws of Physics which is usually not included in Computer Science lessons for beginners.
2. REPETITION CONTROL – When working with MOTION-LIST/PAGEs, the user learned how to use the COUNTING LOOP (see Fig 4.18) and the ENDLESS LOOP (see Fig. 4.21). Only the CONDITIONAL LOOP (WHILE loop) is not implemented in the MOTION tool.
3. SELECTION CONTROL – This concept is introduced to the reader in a "sneaky" way via the use of the EXIT MOTION UNIT by way of the buttons STOP (F8) and EMERGENCY STOP (F9) (see Fig. 4.20).

Thus, the author claims that it is reasonable to associate the concept of MOTION PROGRAMMING with the MOTION tool. If the reader happens to be an instructor or teacher, he or she may be interested to know that the author had noticed among his robotics students that their MOTION PROGRAMMING skills are independent from their COMPUTER PROGRAMMING skills, and to be a well-rounded "roboteer", students would need competency in both sets of skills.

Chapter 5: Using MINI Mobile App

With the MINI Mobile App, a first-time user can access the ROBOTIS provided "Default" Motion Group (e.g. the "default.mtnx" file) and put the MINI robot through its paces for "edutainment" purposes. It has many utility functions built-in: Actuator Test, Motion Offset, Motion File Setting, Server/Client Settings, just to list a few popular ones. The robotics programming capabilities for this App are limited to:

1. Assigning a "Button", i.e. a touch area on the mobile device, to a user-assigned Motion Page/List via its Motion Index Number.
2. Assigning a particular "Gesture" with the mobile device to a given Motion Page/List via its Motion Index Number.
3. Assigning a Voice Command to the mobile device to activate a given Motion Page/List via its Motion Index Number.
4. Using a Virtual Remote Controller (U-D-L-R-1-2-3-4-5-6 buttons) to access two modes of control: "Soccer" (Defense or Offense) and "Fight" modes.

As of the writing of this chapter, the current MINI Mobile App is V.0.9.26 for the Android version (https://play.google.com/store/apps/details?id=com.robotis.darwinmini) and 0.9.15 for the iOS version (https://apps.apple.com/us/app/robotis-mini/id948481762), and **both are still using the MTNX files**, instead of the MTN3 files as created by the new MOTION 3 component of the R+TASK V.3 application (see Chapter 4). Fortunately, there was NO DIFFERENCE between the formats of an MTNX file and an MTN3 file, thus the author just changed the extension MTN3 from this file "MINI_Custom.mtn3" (from Chapter 4) to MTNX to obtain the file "MINI_Custom.mtnx" which can then be used by the current MINI App directly. The interested reader can use Windows NOTEPAD to open and read inside these MTNX/MTN3 files to see the details in ASCII text (they are XML files in actuality). Perhaps in a future update of the MINI App, it can read MTN3 files directly.

The MINI App is well documented at these web links: http://emanual.robotis.com/docs/en/software/mobile_app/mini_app/ and http://emanual.robotis.com/docs/en/edu/mini/, thus the author did not see the point of rewriting these informational details back into this Chapter 5. **Thus, the goal of this Chapter is to help the reader go through a practicum in using the MINI App to add a new button with the author's provided file "MINI_Custom.mtnx".**

5.1 Transferring "MINI_Custom.mtnx" to Mobile Devices

For the purpose of writing this Chapter, the author used either an Android tablet (Samsung Tab A) or an iPad as the mobile platforms.

If the reader uses an Android device and the file "MINI_Custom.mtnx" file is on a Windows PC, then Windows Explorer can be used to drag and drop this file from the PC into the appropriate "ROBOTIS MINI" folder on the Android tablet (see Fig. 5.1).

5.2 Connecting to Bluetooth & Performing Actuator Test

This web link shows how to connect the MINI robot to the MINI App via Bluetooth (http://emanual.robotis.com/docs/en/edu/mini/#connecting-the-robot-with-the-app-using-bluetooth).

Next, it would be a good idea to use the "Actuator Test" function to check out the overall assembly of the servo motors to see if they were at the right locations on the robot's frame and if they were controllable from the reader's mobile device (http://emanual.robotis.com/docs/en/edu/mini/#assembly-check-using-the-

app). A minor note to remember that the current MINI App can only test up to 16 XL-320s on the MINI robot, as the author's robot had 17 servos, he had to test the 17th servo using the MANAGER tool (see Section 2.1.2).

5.3 Associating "MINI_Custom.mtnx" to MINI App

The MINI App by default uses the Motion File "default.mntx", and as we want to use the "MINI_Custom.mtnx" Motion File, we have to modify the "Motion File Settings" which can be found under the Setting icon located near the top right corner of the App's screen when the MINI App is activated (see Fig. 5.3).

5.4 Assigning a New Button for Motion Page/List #45 "Crouching Bows"

In this section, we'll practice how to assign a new "Button" (i.e. a Touch Area on the mobile device) to the Motion Page/List named "Crouching Bows" in the "**CT Custom Motion Group**".

This web link (http://emanual.robotis.com/docs/en/edu/mini/#control-with-buttons) gives good information about how to Add/Edit/Delete/Arrange Buttons inside the MINI App – the reader is recommended to review these steps in this web link first before reading on.

Once connected to the real robot, the MINI App main screen looks like what is shown in Fig. 5.5, whereas the user can either choose to "Run" or to "Edit", but please note that the bottom 3 buttons of the "Run" Screen, i.e. "Stop Motion", "Init Pose" and "Stand Up", are not editable (for further information please review this web link http://emanual.robotis.com/docs/en/edu/mini/#operation). **Furthermore, there is a typo in the right-most label: it should have been "Get Up" (for Motion Unit #2) instead of "Stand Up" which corresponds to Motion Unit #4.**

Fig. 5.5 Main Screens for the MINI App.

5.4 Using Gestures, Voice Recognition, Messenger and Remote Controller

Overall, the MINI App is a good tool to use for beginning robot users but it is too restrictive as a robot programming tool, thus in Chapters 6, 7 and 8, the reader will be shown in-depth and flexible ways to control the MINI robot using the ROBOTIS TASK and PLAY700 tools along with the integration of selected ROBOTIS sensors to the OpenCM9.04-C, as well as other third-party and open-source software such as EDBOT and OPENCV.

Chapter 6: Using TASK

To reach the broadest range of readers, the author will assume that the reader is a completer beginner in the areas of computer programming and robot control, thus Chapter 6 would start with broad robot control concepts before getting into the finer details of programming the MINI robot with the TASK component of the R+TASK V.3 tool.

The R+TASK V.3 tool offers most of the standard programming structures found in other languages such as C/C++. Below is a short description of the most useful features for a beginning programmer (more details on specific items will be provided at appropriate sections in this chapter):

1) ASSIGNMENTS – TASK provides three types of assignment statements: LOAD, COMPUTE and POLYNOMIAL.

The syntax for LOAD is "A = B" with the usual meaning of "Operand A is assigned the Value of Operand B".

The syntax for COMPUTE is "A = B *Op* C" where the RHS represents the Value obtained when performing the "*Op*" operation between Operands B and C. The "*Op*" operations supported are the 5 basic arithmetic operators "+, - , * , / , and % (remainder operation)", and the 2 bit-level operators such as "&" (AND) and "|" (OR). Only INTEGER arithmetic is supported. VARIABLE parameters are supported but arrays are not.

POLYNOMIAL is "format-free". It is the usual "expression" in standard computer languages but it can only include the arithmetic and logical operators listed in the previous paragraph and of course the parentheses "(" and ")" are used to group sub-expressions as needed.

2) LABEL and JUMP statements are supported.

3) CONDITIONS – For Conditional statements, TASK provides the usual logical operators: "&&" and "||" respectively for logical AND and logical OR operations, and also "==", "!=" , "<", "<=", ">", ">=" for equality and inequality tests. The standard IF, IF-ELSE-IF, ELSE structures are supported, but nested conditional statements using parentheses are not supported (so the user will have to apply DeMorgan's theorems to expand complex logical statements as needed).

4) LOOPS – for Repetition statements, TASK provides "WAIT WHILE", "LOOP WHILE", "LOOP FOR", "ENDLESS LOOP" and "BREAK LOOP". Standard computer languages also offer a DO-WHILE loop structure which is not supported in TASK.

5) FUNCTION definition and calling are provided in TASK. A special CALLBACK function can also be used: it is triggered by a hardware timer every 7.81 ms independently of the main TASK program. CALLBACK can only handle up to a maximum of two external device calls.

6.1 Learning TASK Programming Basics

The user is recommended to first view Video 6.1 for a quick working tour of the TASK V.3 application and to learn about the editing basics for TASK V.3 component by following the author's steps in creating the example TASK program called "M_LEDControl_Servo6.tsk3".

The following sub-sections, 6.1.1 through 6.1.5, were written for a "complete beginner" user in mind, but they could be used as "review materials" for readers with more experiences in computer programming from elsewhere.

6.1.1 LED Control with Timer

Video 6.1 just showed the mechanics of creating a TASK program such as "M_LEDControl_Servo6.tsk3" without much explanations for why certain statements are needed and why a certain order of these statements was used. In this sub-section 6.1.1 each statement used will be explained in more details (see Fig. 6.1).

6.1.2 Sequential Control of Servo Positions

The goal of "M_SequentialControl_Servo6.tsk3" is of course to learn the effect of SEQUENCE CONTROL on the motions performed by Servo-6 (see Fig. 6.2):

```
START PROGRAM
{
  // Overall robot initialization
  ID[All]: Torque ON/OFF = FALSE (0)
  ID[All]: LED = FALSE (0)
  // Special initialization for Servo 6
  ID[6]: Torque ON/OFF = TRUE (1)
  ID[6]: Operating Mode = 2
  ID[6]: Torque Limit = 1023
  ID[6]: Goal Velocity = 1023
  // Command Servo 6 to move to 0 degree position
  ID[6]: Goal Position = 512
  WAIT WHILE ( ID[6]: Is Moving == TRUE (1) )
  // Command Servo 6 to move to -90 degree position
  ID[6]: Goal Position = 205
  WAIT WHILE ( ID[6]: Is Moving == TRUE (1) )
  // Command Servo 6 to move to +90 degree position
  ID[6]: Goal Position = 819
  WAIT WHILE ( ID[6]: Is Moving == TRUE (1) )
}
```

Fig. 6.2 TASK program "M_LEDControl_Servo6.tsk3".

6.1.3 Endless Loop Control of Servo Positions

The "M_EndlessLoopControl_Servo6.tsk3" program showcases the use of an Endless Loop with Position Control commands within it (see Fig. 6.3). It uses the previous Servo Control concepts described in Section 6.1.2.

6.1.4 Counting Loop Control of Servo Positions

The "M_CountingLoopControl_Servo6.tsk3" program illustrates the use of a Counting Loop (labeled as LOOP-FOR in TASK program nomenclature). This type of loop is used when the programmer knows exactly how many times that a group of statements needs to be executed by the robot's controller to accomplish a given task (see Fig. 6.7 and Statements 14 through 27).

6.1.5 Conditional Loop Control of Servo Positions with Remote Controller

The TASK program "M_ConditionalLoopControl_Servo6.tsk3" has two goals: the first one is to illustrate the proper use of the LOOP-WHILE structure, and the second one is to introduce the usage of the Virtual Remote Controller RC-100 (see Fig. 6.8).

6.2 Structure of a complete TASK program

In this Section 6.2, the TASK program "M_RC_Walker_Basic_a.tsk3" is used just to illustrate the various components of a typical TASK code. A detailed procedure will be provided in Section 6.3 to illustrate a general approach to solve a typical robotics project resulting in such a TASK program.

Please note that the program "M_RC_Walker_Basic_a.tsk3" uses the "RC_Walk_1" MOTION-GROUP from the "MINI_Custom.mtn3" file (see Fig. 6.10). The reader needs to make sure to download this MOTION-GROUP before downloading and running the "M_RC_Walker_Basic_a.tsk3" code.

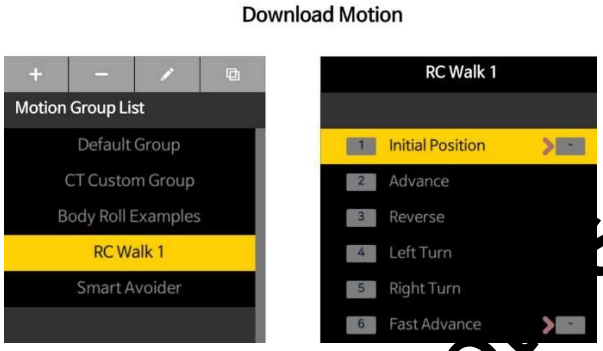

Fig. 6.10 Accessing "RC_Walk_1" MOTION-GROUP.

Fig. 6.11 shows Part 1 of the program "M_RC_Walker_Basic_a.tsk3".

```
1  // MINI RC Walk V. 1
2  // Walk in only 1 direction U-D-L-R at any one time using IF-ELSE-IF
3  // All Rights Reserved - CNT Robotics LLC 2019
4
5  START PROGRAM
6  {
7     // Overall robot initialization
8     CALL Init
```

Fig. 6.11 Part 1 of "M_RC_Walker_Basic_a.tsk3".

6.2.1 Comment & Command Statements

6.2.2 Start & End Commands

6.2.3 Execute Statements

The most common type of EXECUTE Statements is the LOAD type and it is used throughout this example program at Lines 18, 22, 27, 32, 37, 45 to 50 in Figs. 6.12 and 6.13. EXECUTE Statements affect the informational data flow/status from sensors/parameters and the subsequent robot's actions. Without them, the robot would be a "no-fun" static object.

6.2.4 Loop Statements

Only two types of LOOP Statements are used in this example TASK code and they are of the type ENDLESS-LOOP (Lines 10 in Fig. 6.12) and WAIT-WHILE (Line 16 in Fig. 6.12 and Line 51 in Fig. 6.13). Section 6.3 will show that the ENDLESS-LOOP is a direct consequence of the use of the "Sense-

Think-Act" paradigm. LOOP Statements allow the creation of compact code blocks that can be conditionally repeated instead of having to lay them out sequentially in the controller's memory space and thus make more efficient use of this limited resource.

6.2.5 Condition Statements

Only two types of CONDITION Statements are used in this example TASK code and they are of the type IF (Lines 13 and 19 in Fig. 6.12) and ELSE-IF (Lines 24, 29 and 34 in Fig. 6.12. Section 6.3 will show that the CONDITION Statements flow from the use of the EVENT-PROGRAMMING approach. In a way, CONDITION Statements allow the mapping of all possible event/action pairs that are currently considered by the programmer for the robot to deal with. CONDITION Statements allow the robot to respond to a changing environment, in the real world, that it must work within.

6.2.6 Function Statements

The only FUNCTION Statements used are the FUNCTION (definition) used at Line 43 of Fig. 6.13) and the (function) CALL at Line 8 of Fig. 6.11. FUNCTION Statements allow the programmer to use a top-down and modular approach to develop an efficient solution to a given robotics project with defined goals and tasks (see Section 6.3).

6.3 Beginner's Approach for Robotics Projects

In this section, the author strives to provide beginning "roboteers" with a general approach to thinking about robotics projects, as well as concrete sequential steps that ultimately result in an efficient TASK program that can perform the tasks/behaviors that are required of the MINT robot for a given project.

6.3.1 "Sense-Think-Act" Paradigm

Most if not all my beginning robotics students were surprised when I started my robotics short courses by mentioning that they had been doing "Robotics" in their everyday activities and doing it very well too, all by themselves. I used Fig. 6.14 to explain how humans use our Senses to let us know about the external World (i.e. Perception) and apply those Sensations into our Cognitive process (i.e. Thinking) to devise proper Actions/Reactions to the situation at hand. Those Actions/Reactions in turn would change some aspects of the external World resulting in new Sensations triggering the next World-Human-Interactions cycle and so forth.

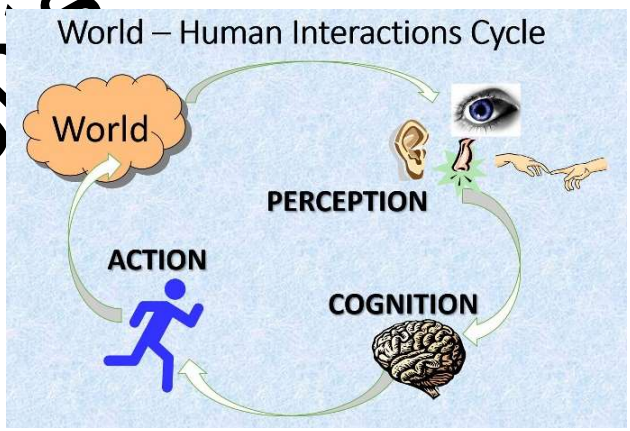

Fig. 6.14 World-Human Interactions Cycle.

6.3.2 Reactive-Control Approach

Next, let's spell out the **goals** for the current project "RC Walker Basic" that can lead to the example code "M_RC_Walker_Basic_a.tsk3" which used MOTION-LISTs/PAGEs defined in the MOTION-GROUP "RC_Walk_1" (see Fig. 6.10):

6.3.3 Event-Action Pairs

With the "Reactive Control" approach, the **first step RC1** is to translate each project goal from the above list into a specific Condition/Event matched with an appropriate Action for the robot to do (see Fig. 6.19).

6.3.4 Basic Remote-Control Concepts (revisited)

Although some Remote-Control concepts were introduced in Section 6.1.4, let's now look more formally at how the OpenCM9.04-C can receive data from the VRC (i.e. the PC) via the BT-210 receiver:

6.3.5 Sensor-Actuator Pairs

The **second step RC2** is to build upon the results of the previous step RC1 by mirroring its Event-Action pair into a matching Sensor-Actuator pair as shown in Fig. 6.20. **The goal is to specify the kind of Sensors/Actuators (in a general sense) to be used and to prescribe how they are to be used.**

6.3.6 Event-Programming Approach

Another benefit for using Event-Action and Sensor-Actuator pairs is that they will help with the actual coding (as shown later in Section 6.3.7), as the TASK syntax would support practically a one-to-one correspondence of all Event-Action pairs A to Z (see Fig. 6.21) into similar "physical" locations within the final program, except for two aspects that had not been yet discussed:

6.3.7 Application to Basic RC Walker

Recalling the previous discussions made in Section 6.3.3, this project deals with **mutually exclusive Event-Action Pairs**, and TASK has a CONDITION structure that will function in this manner. It is called the IF-ELSE-IF structure which can be laid out in pseudo code below:

If (Event 1 is TRUE) then (Play Robot Motion 1)

Else If (Event 2 is TRUE) then (Play Robot Motion 2)

….

Else If (Event 5 is TRUE) then (Play Robot Motion 5)

In "Computational Thinking" practice, these steps would also correspond to the "Algorithm" (Refining) phase.

In practice, diligent users would soon realize that backtracking various steps of this procedure for improving one's TASK program will be the "norm" for any meaningful robotic project.

6.3.8 TASK Support Utilities

The TASK component of TASK 3 also provide several support utilities during the programming and debugging phases of the user's robotics projects:

1. In its Programming Menu (F2) (see Fig. 6.24):
    a. The usual editing functions are implemented – Cut, Copy, Paste, Redo & Undo.

b. The user can export a TSK3 program into several PNG files depending on the length of the TSK3 program or into a single ASCII text file.
c. The user can also choose the appropriate COM port before downloading one's TASK code.

2. In its Debugging Menu (F3) (see Fig. 6.25):
   a. The user can start or stop the Program Output Monitor to see print outputs at runtime, if the user's program uses PRINT or PRINTLN commands.
   b. If the user's program uses the Remote Controller RC-100 features, then the Virtual Remote Controller facility is accessible here. The user needs to remember that the arrow keys (U-D-L-R) and the "number" keys near the top of the keyboard can also be used instead on left-clicking with the mouse into the "buttons" shown in Fig. 6.25.

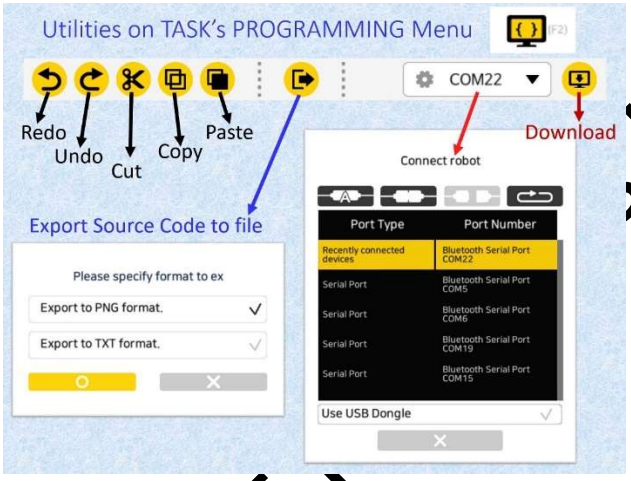

Fig. 6.24 Utilities found in TASK's Programming Menu (F2).

6.4 RC Walker with Independent Servos Control

Expanding on the previous project "Basic RC Walker", let's add a new capability to our MINI. Instead of using only 1 RC button at any time to control the robot at runtime, let's now use multiple buttons simultaneously on the RC-100 (Physical or Virtual) to accomplish these tasks:

1. We want to use "U-D-L-R" buttons to serve as a **Walk Direction** (only one direction is allowed at any time).
2. The buttons "1" and "3" are to be used when we want to **raise ("1") OR lower ("3")** the **right arm** of the MINI via **direct control of Servo ID=1 (i.e. right shoulder joint) while the robot is walking in any direction or standing still**.
3. The buttons "2" and "4" are to be used when we want to make the MINI's **head scan left ("2") OR scan right ("4")** via **direct control of Servo ID=17 while the robot is walking in any direction or standing still**.

This new capability will require us to resolve two issues:

1. How to separate the information from the different RC buttons from the single message (e.g. "DataIn") that our program would receive from the VRC when the user pushes on several RC buttons at the same time (see Section 6.4.1).

2. From Chapter 4, we know that when a MOTION-LIST/PAGE is being played all 17 servos are controlled by this MOTION-LIST/PAGE, thus how can a TASK program be written to control only specific servos among those 17 servos? (see Section 6.4.2).

6.4.1 Filtering a RemoCon Message

So far (Sections 6.1.5 and 6.3), the reader has been shown the basic usage of messages sent by the TRC, let's now look closer at the structure of these "RemoCon" messages.

A full "RemoCon" packet is based on a 16-bit user-designed message wrapped up in a 6-byte packet (http://emanual.robotis.com/docs/en/parts/communication/rc-100/#communication-packet). For this Section 6.4, we'll restrict ourselves to using only the lower 10 bits of this 16-bit message (in Section 6.7, the reader will be shown how to use all 16 bits).

6.4.2 Servo Joint Offset Special Value "1024"

The last time that we used Servo Joint Offset parameters was in Section 4.4 regarding the OFFSET-POSE being used to adjust for variations in the geometric framing configuration of a typical "physical" MINI robot, as compared to the ideal one represented by the "graphical" MINI robot. These values were small, actually they are restricted to the range from -255 to 255 (please see more details at this web link http://emanual.robotis.com/docs/en/software/rplus1/task/programming_02#joint-offset). This link also mentions that a special value of "1024" can be used to spare specific servo(s) from the effect of a robot motion being performed.

6.4.3 Construction of "M_RC_Walker_IS.tsk3"

As this is another RC application, the general algorithm developed for "M_RC_Walker_Basic_b.tsk3" should still be applicable. In the "Basic" application only the "Walk Direction" was needed: i.e. we need only to know which U-D-L-R Button is pushed and then play the appropriate MOTION-LIST/PAGE to make the MINI walk in the commanded direction.

This "Independent Servo Control" project also illustrates different ways to use IF, ELSE and ELSE-IF structures to check for events that are critical to the robot's behaviors at runtime and the main challenge was how to manage those structures within the Main Endless Loop. The "management" approach that the author used was to change the focus from "trees" to "forest" and vice-versa to make sure that the "Sense-Think-Act" Paradigm is adhered to at all levels of abstraction, along with the use of Flag/State Parameters to allow the programmer to keep track of the different "robot-states" that may be involved.

6.5 RC Smart Avoider

In this project "M_RC_Smart_Avoider.tsk3", Remote Control techniques are combined with Autonomous Behaviors based on Sensor Input via two NIR distance sensors (IRSS-10) mounted on either side of the MINI's head piece (see Fig. 1.4).

As this project requires fast responses from the IRSS-10 sensors as well as from MOTION-LIST/PAGEs being performed, there are two new concepts that readers need to master: programmed stoppage of a current MOTION-LIST/PAGE being performed and appropriate use of the CALLBACK Function.

6.5.1 "Programmed Stoppage" of Motion-List/Page

In the MOTION-GROUP "RC_Walk_1" used so far (see Fig. 6.10), each of its MOTION-LIST/PAGE performs 3 repetitions of the chosen robot movement such as Advance, Reverse, Turn Left or Turn Right for

each time that it is invoked. For this "Smart Avoider" project where a quick response to an obstacle just detected is of paramount importance, we will need to use a different MOTION-GROUP where robot movements are shorter in time duration and performed only once when invoked. Thus the "Smart Avoider" MOTION-GROUP was created (see Fig. 6.40).

6.5.2 CALLBACK Function

The TASK Sub-Tool offers a special Function named CALLBACK which is executed independently of the Main Program and its User-Defined Functions. It executes itself every 8 ms (7.81 ms, to be exact) which corresponds to the smallest time period allowed between KEY-FRAMES defined in a MOTION-UNIT (see Section 4.1.2). Because of the short execution time of 8 ms, the CALLBACK Function has some restrictions on its size and the types of commands that can be used in it:

- Its size cannot be more than 512 bytes.
- LOOP, LABEL, JUMP and CALL types of commands are not allowed.
- A maximum of 2 External Device Calls (e.g. access to Sensors, RC-100 buttons or Dynamixels) are recommended. More programming tips for accessing Dynamixels are presented in Section 6.8.4.

6.5.3 Construction of "M_RC_SmartAvoider.tsk3"

For this project, the "Sense-Think-Act" still works well for us, however the Event-Action pairs become more complex in structure and intertwined as the robot's tasks get more sophisticated. The author's approach had been to create the **Essential Algorithm** first (i.e. the "forest" view) and to deal with the **Linkages** as they arose (i.e. the "tree" view).

6.5.4 Alternate Solution for Smart Avoider

There is an alternate algorithm/solution to the Main Endless Loop for the "Smart Avoider" project as shown below:

6.6.2 New "Walk Execute" Approach

The "Essential Algorithm" for the "Random Walker" project is a modification of the Main Endless Loop used in "M_RC_SmartAvoider_alt.tsk3" and it is shown below:

6.7 The "Mimicking MINIs"

Back in Section 6.4.1, the reader had been shown how to filter a "Remocon" Message to obtain only specific information about specific bits comprising the RC-100 message which has only 10 bits, each bit corresponding to each of the 10 Buttons U-D-L-R-1-2-3-4-5-6. However, a full "Remocon" message has 16 bits (http://emanual.robotis.com/docs/en/parts/communication/rc-100/#communication-packet), thus this "Mimicking MINIs" project will show the reader how to use the other 6 bits via a "Message Shaping" procedure.

For this project, **the reader does need to have access to two MINIs**, one will be set up as the Leader and the other as the Follower. The servos of the upper body of the Leader (i.e. Servo IDs from 1 to 6 and 17) will be disabled so that they can manipulated by hand. **The goal of this project is to design TASK codes to run on the Leader and Follower robots such that when the Leader's disabled servos are manipulated one-by-one or as a unit, the corresponding servos on the Follower will match exactly their movements.**

6.7.1 Master & Slave BT-210

First, the BT-210s on the Leader and Follower robots need to be modified so that **they communicate only to each other** (and not to the PC or Mobile Device as we had been using them). In wireless communications speak, the Leader's BT-210 needs to be set up as a Master and the Follower's BT-210 needs to be set up as a Slave. The ROBOTIS e-manual does carry how-to instructions for this procedure, along with a video, at this web link http://emanual.robotis.com/docs/en/parts/communication/bt-210/#communication-mode.

6.7.2 "Message Shaping" Concept

For this project, the Leader robot needs to send to the Follower robot information about its own chosen XL-320 regarding its **ID** number (valued at 1 through 6 or at 17) and its corresponding **Present Position** (i.e. a number between 0 and 1023). When the Follower receives these items, it can use them to match its own servo **ID** and to set its **Goal Position** with the Leader's **Present Position**.

6.7.3 "Open-Loop" Solution for "Mimicking MINIs"

In the "Open Loop" solution, the Leader robot just pumps its information over to the Follower robot which will strive to match the Leader's motions, whether the Follower can do the required motions or not.

6.7.4 "Closed-Loop" Solution for "Mimicking MINIs"

In the "Closed Loop" solution, the Leader robot still pumps its information to the Follower robot which will strive to match the Leader's motions, however if some Follower's servo experiences a Torque Overload situation (because it cannot go where it is supposed to go, for example it gets stuck by some obstacle), then the Follower would send information (to the Leader) about the overloaded servo's ID and its Servo Position where the overload is occurring. This means that "R_Remocon" messages need to go both ways between the Leader & Follower robots. Thus, both Leader and Follower codes for the "Closed Loop" solution will now have a "Sending" section and a "Receiving" section.

6.8 Using MINI's Arms as Gripper

Back in Section 4.4, we encountered the concept of JOINT-OFFSET for a typical Dynamixel XL-320 in the MOTION Sub-Tool and in context of the Calibration Pose for the MINI robot, there we used it as **positive or negative** angular rotation values (i.e. in **degrees**). Then in Section 6.4.2, we used it in the TASK Sub-Tool with the special value of "**1024**" to isolate targeted Dynamixels from the effect of a MOTION-LIST/PAGE being played in order to use GOAL-POSITION commands directly on these Dynamixels (please remember that "legal" GOAL-POSITION values are from 0 to 1023 – see Fig. 1.3). In this section, the reader will see a third way to use JOINT-OFFSET within the numerical range from -255 to 255, and in concert with a MOTION-LIST/PAGE being played. For more information regarding JOINT-OFFSET, please visit this web link http://emanual.robotis.com/docs/en/software/rplustask3/task_parameters/#joint-offset. **Essentially, a JOINT-OFFSET is "added" to the SERVO POSITION defined in a MOTION-UNIT to become the "final" value to be used as GOAL-POSITION for a targeted servo.**

6.8.1 "MOTION from STILLNESS" using Variable Joint-Offset

In this mini-project "M_Head_JointOffset.tsk3", we are using the "Arms Gripper" MOTION-GROUP and the MINI is programmed to play "continuously" the MOTION-LIST/PAGE named "Initial Position", i.e. the Ready Pose. In this "Initial Position" MOTION-UNIT, there is only 1 KEY-FRAME where the

Head Servo (#17) is set to go GOAL POSITION "512" (i.e. head facing forward), thus playing "Initial Position" continuously means to keep the robot in STILLNESS. However, by varying the JOINT-OFFSET of Servo #17 appropriately, we can get the Head Servo into MOTION, that is "MOTION from STILLNESS"!

6.8.2 Application to Adaptive Grasping

We are now ready to apply the above "Joint Offset" technique to the two arms of the MINI to make them function as a gripper that can adapt to different sizes of the object being grasped.

The author implemented two versions of the TASK codes for this "Arms Gripper" project: one uses the Standard Function protocol ("M_ArmsGripper_F.tsk3") while the other uses the CALLBACK Function protocol ("M_ArmsGripper_CB.tsk3").

6.8.3 Standard Function (SF) Solution

Fig. 6.70 shows Part 1 of the Main Program for "M_ArmsGripper_F.tsk3" using the usual initializations in Function "Init".

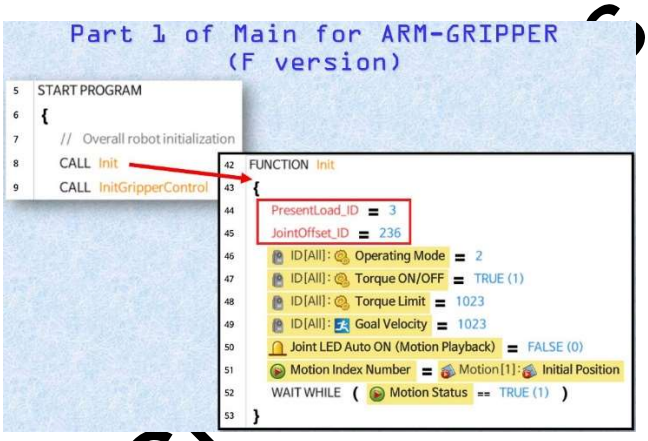

Fig. 6.70 Part 1 of the Main Program for "M_ArmsGripper_F.tsk3" and Function "Init".

6.8.4 CALLBACK (CB) Function Solution

In two previous projects, "IR Smart Avoider" (Section 6.5) and "Autonomous Random Walker" (Section 6.6), we used the CALLBACK Function **to read two IRSS-10 Sensors** which were directly connected to I/O ports 1 and 4 of the OpenCM9.04 Hardware Controller STM32F103CB (see Fig. 6.41). As it took only about 0.02 ms to obtain each sensor reading, as compared to the 8 ms cycle of the CALLBACK Function, we did not have to modify the way we programmed the Main Program and the Standard Functions that were accessing these sensors.

However, with this "Arms Gripper" project, we will need to access 4 XL-320s that are connected to their own Dynamixel network which can be considered as **external** to the STM32F103CB Controller. Furthermore, as the preceding TASK code "M_ArmsGripper_F.tsk3" indicated, the Main Program and Standard Functions would be now accessing the same Dynamixels that the CALLBACK Function would also access, possibly at the same time. Thus, the "real-time" communications time delays between various controllers and the potential network packets collision issues would need to be dealt with, in this CALLBACK solution.

6.9 Autonomous Dowel Scanner using DMS-80

This last project of Chapter 6 uses the Smart Avoider Motion Group and uses the DMS-80 sensor (connected to I/O Port 2) to search for 2 white wood dowels placed at random in front of it, and next to command the MINI to autonomously approach the "closer" dowel first, and then to relocate the remaining dowel before approaching it as the final step.

Fig. 6.84 illustrates the physical layout for the "Dowel Scanner" program "M_DowelScanner.tsk3" as well as its constraints:

1. At the Start Position, the DMS-80 should be "looking" at empty space, as the proposed TASK program uses this condition to establish the "background" level as provided by the DMS-80 sensor readings.
2. At the Left Limit and Right Limit Positions, the DMS-80 should also be "looking" at empty space as a necessary assumption for the Dowel Scan algorithm to work.
3. The two white dowels can be located anywhere between the Left Limit and Right Limit Positions (but not in line with the Start Position). Essentially, "Dowel 1" should be on the robot's Left and "Dowel 2" should be on the robot's Right.

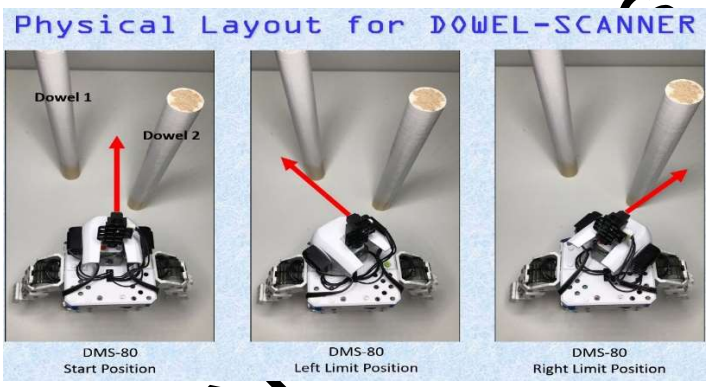

Fig. 6.84 Physical Layout and Contraints for Program "M_DowelScanner.tsk3".

6.9.1 Scan Background & Set Dowel Threshold

6.9.2 Scan & Locate Both Dowels

6.9.3 Relocate & Approach Closer Dowel

6.9.4 Relocate & Approach Remaining Dowel

Chapter 7: Using PLAY700 Mobile App

The Mobile App "R+m.PLAY700" was originally created to accompany the PLAY700 kit (http://www.robotis.us/software/play700/), but it also works with the OpenCM9.04-C using SMART DEVICE commands in conjunction with SMART CONSTANT parameters inside the TASK tool (see Fig. 7.1). Detailed information about these SMART functions and their corresponding arguments are contained in the file "SmartDeviceControlTable.PDF" accessible at www.cntrobotics.com/mini-book. This PDF file is a Chrome-translated English version of this Korean web page http://support.robotis.com/ko/software/mobile_app/r+smart/smanrt_manual.htm#Actuator_Address_0B3.

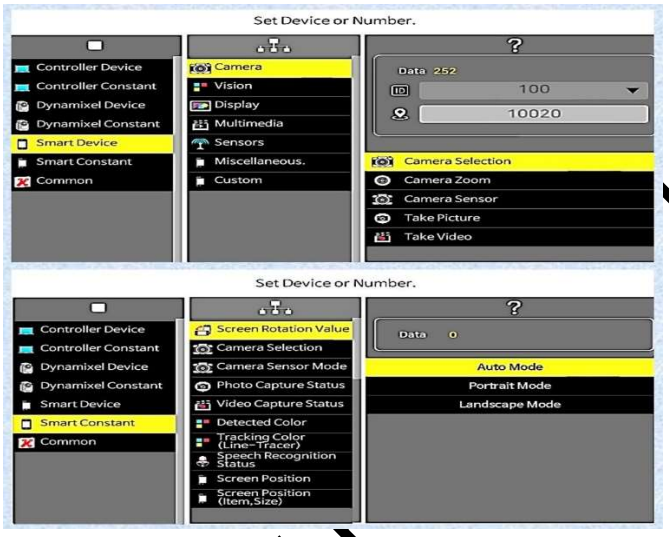

Fig. 7.1 SMART DEVICE & CONSTANT Options inside TASK Sub-Tool.

The R+m.PLAY700 App allows users to add more functionalities to standard TASK programs (see Chapter 6) by leveraging existing mobile services such as graphics, multimedia, video camera or speech recognition.

The goal of this Chapter is to present readers with a firm foundation in using the R+m.PLAY700 App with R+TASK V.3 via several targeted projects.

7.1 Installation of R+m.PLAY700

The reader can install the latest version of R+m.PLAY700 on iOS or Android devices at the appropriate web links (https://play.google.com/store/apps/details?id=com.robotis.play700&hl=en or https://itunes.apple.com/us/app/play700/id1156037721?mt=8).

7.1.1 File Structure for PLAY700 App

For development work, the author uses a Windows 10 PC which interfaces well with Android devices, and all file and folder management can be done via Windows Explorer. Fig. 7.2 shows the folder/file structure for PLAY700 on an Android device.

7.1.2 Settings for PLAY700 App

Fig 7.4 shows a screenshot of the main menu window for the PLAY700 App where the reader can see a "gear' icon on the top right corner. Once this "gear" icon is tapped once, the "Settings" screen is shown as in Fig. 7.5.

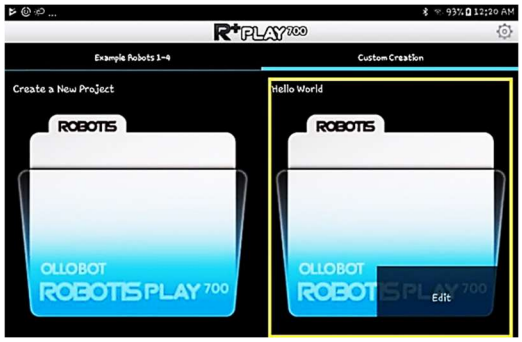

Fig. 7.4 Main Menu of PLAY700 app, opening on Custom Project "Hello World".

7.2 PLAY700 Project Features

Going back to Fig. 7.4, when the user taps on the "Edit" button of a project, a screen as shown in Fig. 7.6 displays all the components/tools that can be used for this project. The current Android version (0.9.5.1) has all these tools operational, **but for the current iOS version (1.0.7), the following six tools are not yet functional**:

"Instrument" in the Multimedia group.
"Illumination" in the "Sensor" group.
"Received SMS", "Status Bar", "Vibration", "Application" in the "Other" group.

7.3 "Hello World" Project

There is an extensive list of "SMART DEVICE" commands as shown in Fig. 7.1 and this Section 7.3 should provide readers with a basic usage of the tool chain TASK/R+m.PLAY700. This web link http://emanual.robotis.com/docs/en/software/plustask3/task_parameters/#smart-device has more information about the Smart Devices supported in TASK V.3.

This project will show how to display the text object "Hello World!" on the Mobile Device's Display Screen in different sizes and colors, and at randomized screen locations.

As far as TASK is concerned, the Mobile Device's Display Screen is divided into 25 zones arranged in a 5x5 grid in either Portrait or Landscape mode (see Fig. 7.7).

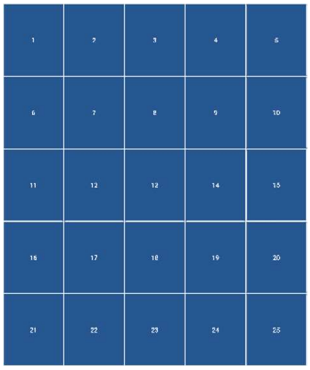

Fig. 7.7 Organization of the Display Screen on Mobile Device vis-à-vis TASK Tool.

7.4 "Touch RC Walker" Project

This "Touch RC Walker" project's goal is to adapt the robot functionalities described in the "RC Walker with Independent Servos Control" project (i.e. "M_RC_Walker_IS.tsk3" in Section 6.4) using selected Smart Device features such as "Touch Area", "Text Display" and "Instrument Play" as well as a new operational feature which is the capability to change the Goal Velocity of all the servos of the MINI. This project uses the "RC Walk 1" MOTION GROUP from the file "MINI_Custom.mtn3".

7.4.1 "Touch Areas" in R+m.PLAY700 App

So far we have only used the RC-100 with its 10 buttons as a Remote Device for controlling the robot, and each Remocon message contains ON/OFF information regarding these 10 buttons "individually", so we have developed programming techniques to deal with that operational feature (see Section 6.4). However, the R+m.PLAY700 App handles the Mobile Device's Display Screen quite differently as it divides this Screen into 25 Touch Areas arranged in a 5x5 grid in either Portrait or Landscape mode (see left picture in Fig. 7.12). Furthermore, this App can only report to a TASK program about 2 Touch Areas at any time during the runtime of a TASK/PLAY700 project. In other words, the programmer/operator can use only up to two fingers at a time when using the Mobile Display as a Remote Controller (as compared to all 10 buttons on the RC-100). But the programmer will be able to distinguish which "finger" touches the screen first as "Touch Area 1", and the later finger-press as "Touch Area 2". Inside the TASK Sub-Tool, these "Touch Areas" are accessible via the "Smart Device" panel and in the "Sensors" category (see right and bottom pictures in Fig. 7.12). The Parameters "Touch1" and "Touch2" would contain a numerical value in the range from "1" to "25" depending on which Touch Area(s) on the Mobile Display get pressed, and they would contain "0" if all fingers are off the Mobile Display. Consequently, a different interface programming technique needs to be developed to handle Touch Areas (see Section 7.3.4).

Communications wise, the "SMART: Touch Area 1" and "SMART: Touch Area 2" commands are READ instructions that are sent from the MINI Controller to the Mobile Device through the 57 Kbps Bluetooth interface. Once the Mobile Device receives these instructions, it uses its own services to determine the current status of its Touch Areas and reports this information back to the MINI Controller via Bluetooth. Thus, this is a lengthy process as compared to the reading of Buttons on the RC-100.

Fig. 7.12 Accessing & Programming Touch Areas on Mobile Display.

7.4.2 "Text" Items in R+m.PLAY700 App

For this Touch RC project, the first 10 "Text" Items as shown in the left picture in Fig. 7.13 are used. These "Text" Items need to be registered separately in an appropriate PLAY700 Project before using them in the TASK program (see Part 1 in Video 7.2).

Fig. 7.13 also show how to use the "SMART: Text Display" command to display a specific "Text" item on the Mobile Display at runtime whereas the programmer needs to specify:

- Its display "Position" within the 5x5 grid of "Touch Areas" described earlier.
- Its numerical "Index" as registered on the PLAY700 App "Text" utility.
- Its display font "Size" [0-255].
- Its display "Color" from 10 choices: Unknown, White, Black, Red, Green, Blue, Yellow, Light Gray, Gray and Dark Gray.

These options are combined into a 32-bit integer value using the protocol described in Section 7.3 and then assigned to the "SMART: Text Display" utility which runs on the Mobile Device (see Fig. 7.14).

7.4.4 "Instruments Play" in R+m.PLAY700 App

The PLAY700 App also plays musical scales on 128 Musical Instruments (from "Acoustic Grand Piano" [1] to "Gunshot" [128]. For each instrument, there are 10 Octave settings [1-10] and 12 Musical Scales or Notes [1-12]: e.g. Note 1 is "Do", while Note 3 is "Re", and so forth until Note 12 which is "Shi". A 3-byte SMART CONSTANT is associated with the Musical Instruments service where the lowest byte (Byte 1) contains the Note's value, while the next higher Byte (Byte 2) contains the Octave's value and Byte 3 contains the Instrument's value. The 4th byte of a typical SMART CONSTANT is not used for the Musical Instruments service and needs to be set to zero (see Fig. 7.15).

7.4.5 Construction of "M_TouchRC_Walker_IS_SD.tsk3"

This program "M_TouchRC_Walker_IS_SD.tsk3" uses the overall algorithm of the previous project "M_RC_Walker_IS.tsk3" which is described in details in Section 6.4.3. Thus, only relevant differences due to the use of SMART DEVICE functions will be emphasized in this section.

Fig. 7.16 describes the initialization steps done for servos in "M_TouchRC_Walker_IS_SD.tsk3" which are similar to the ones used in Fig. 6.34 for "M_RC_Walker_IS.tsk3", except for the setting of the Servo "Goal Velocity" Parameter:

- In Fig. 6.34, the "Goal Velocity" for All servos was set to a fixed constant value of "1023" (Line 130).
- In Fig. 7.16, the "Goal Velocity" for All servos was set to a Variable named "Speed" (Line 150) which was initialized to "800" in the previous Line 149.

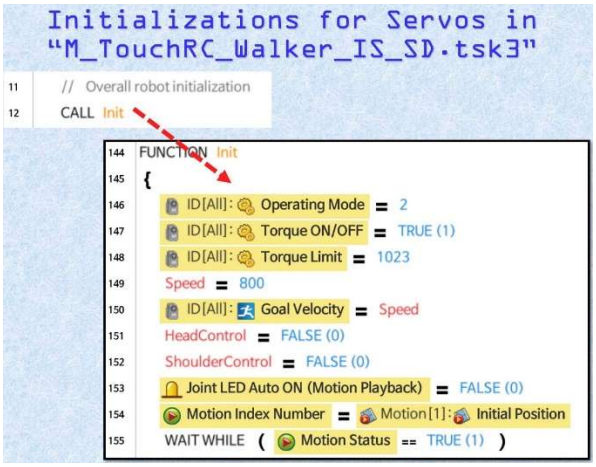

Fig. 7.16 Initializations for Servos in "M_TouchRC_Walker_IS_SD.tsk3".

Fig. 7.17 describes initialization steps done for the SMART DEVICE (i.e. R+m.PLAY700 App on the Mobile Device):

- Line 158 sets the Mobile Display in Portrait Mode (i.e. a SMART Write command).
- Line 159 clears the "Text" Display Screen by sending a "0" to the PLAY700 App (another SMART Write command).
- Line 161 makes the MINI Hardware Controller **wait until** a "SMART Noise" Parameter coming from the Mobile Device and with a dB value larger than 50 (usually the author whistles loudly into the Mobile Device to achieve this result). **Line 161 is critical to a smooth run-time coordination between the TASK program running on the MINI and the corresponding PLAY700 App running on the Mobile Device**. The reader may remember from Section 1.3.4 that as soon as power is applied to the MINI, its embedded TASK program will immediately run. On the other hand, the human operator will always take some time to start the appropriate Custom Project on the PLAY700 App and the Mobile Device will always take several seconds to connect the Mobile Device Bluetooth port to the BT-210 running the MINI. Essentially Line 161 allows the PLAY700 App to catch up with the much faster TASK program. Without Line 161, all the SMART Text Display commands (Lines 164-176) would be sent over to the Mobile Device when it is not ready to receive them, and therefore would be "lost", i.e. the Mobile Device won't display the "Menu" as shown in Fig. 7.14. **In practice, it is best to make sure that the appropriate PLAY700 Project is activated before turning on the MINI to start the TASK program.** *A side note for the interested reader: on newer ROBOTIS controllers such as the CM-550, there is a "Check Smart-Device Connection" TASK command that can be used to check for the actual BT connection between the Mobile Device and the MINI. This command can be used before issuing such Lines as 158-159 for best synchronization between these two devices.*

Fig. 7.17 Initializations for SMART DEVICE in "M_TouchRC_Walker_IS_SD.tsk3".

- Lines 164-176 produce the "Menu" as shown in Fig. 7.14. *A sharp reader may have noticed that certain Text Items were NOT "colored" correctly: "Slow", "Fast", "Raise A", "Lower A", "Scan L" and "Scan R". This was just because the author's Samsung Tab A 8" was not 100% compatible with the PLAY700 App at the time of writing of this book!*

7.5 "Mixed Control Walker" Project

For this "Mixed Control" project, the primary goal is to have two ways of controlling the Moving Directions of the robot and the Servos Goal Velocity setting using "**Touch Areas**" and "**Speech Recognition**". Secondary goals are to show the usage of the "**Text-to-Speech**" utility and the implementation of a Touch-based **Emergency Stop**. This project also uses the "RC Walk 1" MOTION GROUP from the author's file "MINI_Custom.mtn3".

7.5.1 "Speech Recognition" and "Text-to-Speech" in R+m.PLAY700 App

The "Speech Recognition" Function of the PLAY700 App relies on the web-based Google Speech Engine and it is a lengthy process. Thus, it is designed as a "FIRST FINGER" function.

Both "Speech Recognition" and "Text-to-Speech" functions rely on the same list of "Text" Items used for the "Display" function (see Figs. 7.24 and 7.25).

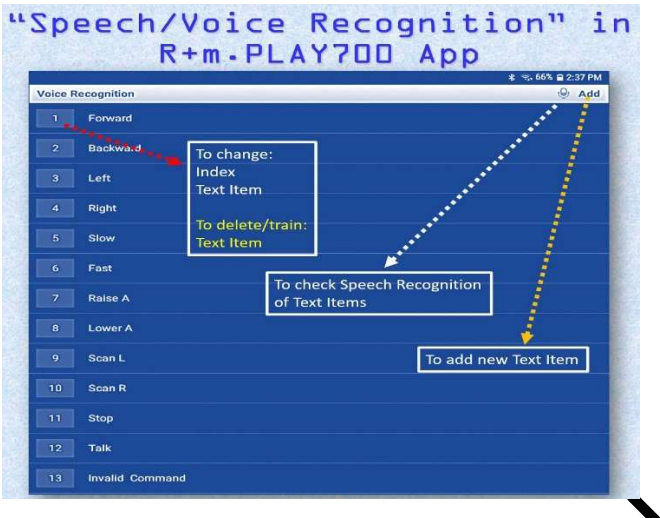

Fig. 7.24 "Speech/Voice Recognition" Utility in R+m.PLAY700 App.

7.5.2 Construction of "M_MixedRC_Walker_IS_SD.tsk3"

Fig. 7.26 shows the Function "Init" for the project "M_MixedRC_Walker_IS_SD.tsk3" which has 3 new statements:

- Line 196 initializes Parameter "InvalidCommand" to FALSE. This parameter is used to report on the success/failure of the Speech Recognition process.
- Line 197 initializes Parameter "VoiceCommand" to "Text" Item 11 which is "Stop" (see Fig. 7.24).

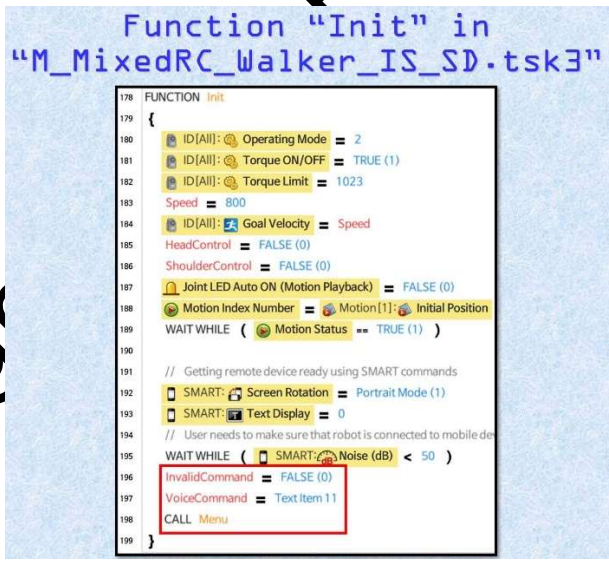

Fig. 7.26 Function "Init" in "M_MixedRC_Walker_IS_SD.tsk3".

- Line 198 calls the Function "Menu" which is listed in Fig. 7.27, showing the Display of two new "Text" Items "Stop" and "Talk" (Line 268 and 269 respectively).

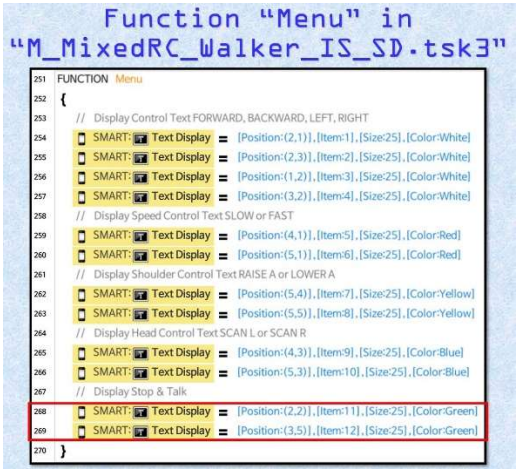

Fig. 7.27 Function "Menu" in "M_MixedRC_Walker_IS_SD.tsk3".

Fig. 7.28 shows that "Text-to-Speech" is used to "accompany" the MOTION PLAY of each Moving Direction performed by the robot when called upon.

The "Speech Recognition" utility (i.e. "Text" Item "Talk") is implemented as a "FIRST TOUCH" function, thus it is added as a new "ELSE-IF" block to the existing "IF-ELSE-IF" structure used for handling the Moving Directions of the robot (as previously shown in Fig. 7.19).

Fig. 7.29 shows the implementation details of "Speech Recognition" in the project "M_MixedRC_Walker_IS_SD.tsk3":

- Line 64 shows the usage of a WAIT WHILE LOOP based on "SMART: Touch Area 1", meaning that a "Press and Release" scheme has been implemented for "Talk" (same approach as used earlier for "Slow" and "Fast" in Section 7.3.5).
- The "Speech Recognition" process is a multi-step procedure illustrated by Lines 66, 67 and 69:
  - Line 66 starts the "Speech Recognition" process (i.e. send a "1" to PLAY700 App). At this point in time, the user would see the "Speak to Me" prompt appear on the Mobile Device. The user then would need to voice out (*as soon as possible and in the same tone of voice and speed as when doing speech-training*) the wanted verbal command among the following "Text" Items: Forward, Backward, Left, Right, Slow and Fast. This process a has a built-in time-out period of about 5 seconds.

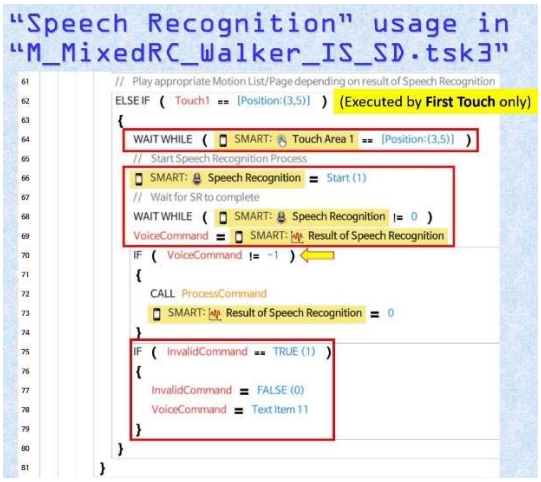

Fig. 7.29 "Speech Recognition" implementation in "M_MixedRC_Walker_IS_SD.tsk3".

  - The "Speech Recognition" tool would then pick up this verbal input and try to match it up with its web database for the actual text as previously created in the PLAY700 "Text" Item list. This process is quite lengthy thus Line 68 is used to wait for its termination. The "SMART: Speech Recognition" command is a READ instruction sent from the MINI to the Mobile Device. When the "Speech Recognition" process is concluded on the Mobile Device, a value of "0" is returned by the Mobile Device to the MINI Controller.
  - It is then OK to save the "SMART: Result of Speech Recognition" value (returned from PLAY700 App) into Parameter "VoiceCommand" (Line 69).

- If Parameter "VoiceCommand" is **valid**, i.e. **not equal to -1**, Function "ProcessCommand" is called (see details in Figs. 7.30 and 7.31) and the Parameter "Result of Speech Recognition" is cleared to 0 (Line 73) upon return from Function "ProcessCommand".
- Lines 75 to 79 handle the situation when an "InvalidCommand" is concluded while running Function "ProcessCommand". When this happens, Parameter "InvalidCommand" is reset to FALSE and "VoiceCommand" is reset to "Text" Item 11 (i.e. "Stop"). At this point, the MINI Controller exits the big IF-ELSE-IF structure handling the "Moving Directions" within the Main Endless Loop.

In this project, an "Emergency Stop" is programmed as a FIRST or SECOND TOUCH option for the operator (see Line 83 in Fig. 7.32):

- Lines 85-90 detail the steps executed next:
  - Line 85 issues a MOTION (-1) command to **immediately stop** the robot current MOTION-LIST/PAGE from being played and have the robot perform the corresponding EXIT MOTION UNIT.
  - Lines 86-90 use a LOOP WHILE to keep on playing "Shi" note on the "Instrument" FX 8, as long as the operator keeps pressing on the Display Screen with ONE or TWO fingers. The music will stop when all fingers are off the Mobile Screen.

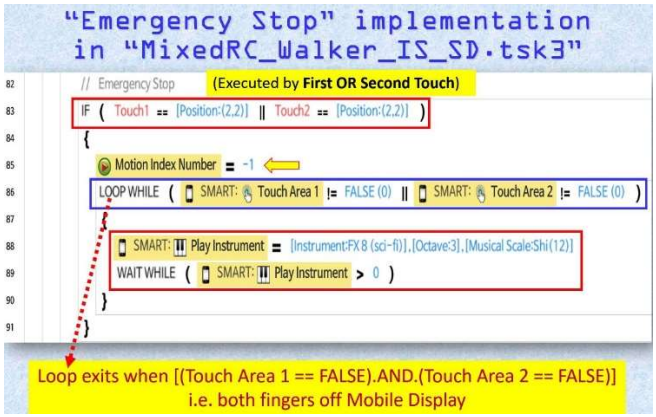

Fig. 7.32 "Emergency Stop" Implementation in "M_MixedRC_Walker_IS_SD.tsk3".

7.6 "Camera Tracker" Project

The reader may remember from Section 4.4 that the author had no success in mounting a mobile phone in front of the MINI robot and maintaining any stable robot motion because of the location of the extra weight of the Mobile Phone. The next option is of course is to mount the Mobile Phone on the backside of the MINI (see Fig. 7.33) for this "Camera Tracker" project.

Fig. 7.33 "Back Mount" of Mobile Phone on Author's MINI.

Fig. 7.33 also shows the JOINT-OFFSETs to be applied to Servos 11, 12, 13 and 14 (essentially the robot's knee and ankle joints) to keep the overall Center of Gravity within the MINI's ground contact area for the Calibration Pose as well as for the Initial/Ready Pose.

With this mounting option, the author could make the robot turn Left and Right in a stable manner (however still no stability obtained for walking Forward or Backward).

Besides taking/recording pictures and videos via the built-in cameras on the Mobile Device, the R+m.PLAY700 App also provides several rudimentary Image Processing modes (see Fig. 7.34):

- In the "Face Detection" mode, the SMART "Face Detection Area" command would return a number between 1 and 25 to the TASK program, representing the Zone Number where the "face" was detected.

- In the “Color Detection” mode, only pixels within Zone 13 (i.e. Screen Center or Position [3,3]) are evaluated for their color values to see if they fit, on the average, certain “fixed” internal numerical RGB criteria for 4 possible color values: 2 for Black, 3 for Red, 4 for Green and 5 for Blue.
- In the “Motion Detection” mode, the SMART “Motion Detection Area” command would return a number between 1 and 25 to the TASK program representing the Zone Number where “Motion” was detected.
- In the “Line Detection” mode, the user first needs to specify which color the App should be tracking for: 2 for Black, 3 for Red, 4 for Green and 5 for Blue. Furthermore, only the top 5 Screen Zones, i.e. Zone 1 through 5, are scanned by the App, thus the SMART “Line Detection Area” command would return a number between 1 and 5 to the TASK program if the user chosen color is found among those zones at runtime. If no such-specified color is found, a value of “0” is returned to the TASK program.

Chapter 8: Using EDBOT™ Tools

So far when we use ROBOTIS software tools, the robot run-time codes always reside on the MINI's OpenCM9.04-C Controller in a "compiled" binary machine code format which is about the best situation for a robot to perform in, along with local access to Dynamixels and Sensors at a communication rate of 1 Mbps (Chapter 6). However, we also saw the limitation of the MINI's firmware as we had to leverage in the PLAY700 App for Multimedia Services, albeit at 57 Kbps (Chapter 7).

In this Chapter 8, the robot control software runs instead on a general-purpose personal computer such as the Windows PC, Mac or Raspberry Pi and inside an "interpreting" shell, meaning that each line of code is "interpreted" one by one at run-time and converted into an appropriate command packet. This packet is then sent via the EDBOT tool to the OpenCM9.04-C through the BT communication hardware at a mere 57 Kbps. Then EDBOT waits for the MINI to respond with Dynamixels and Sensors data which are returned via the same 57 Kbps BT port (see Fig. 1.17). The EDBOT/SCRATCH wait time for Sensors data is around 100 ms, so data throughput is the main issue to consider when programming with EDBOT. However, this throughput issue is traded for the access to a much more powerful and faster computer with a variety of Multimedia resources to tap into. When using PYTHON with EDBOT, more sophisticated programming techniques such as multi-threading and access to software packages like OpenCV are then available to MINI programmers.

In January 2019, SCRATCH 3 (https://en.scratch-wiki.info/wiki/Scratch_3.0) was released with an online interface and a desktop interface (https://scratch.mit.edu/download). In March 2019, EDBOT V.5 was released allowing the additional usage of SCRATCH 3 via a web interface (http://scratch.ed.bot/) – a desktop version may be released in the future. Unfortunately, this web interface yields a slower communications speed with the MINI robot and makes autonomous behaviors harder to achieve, thus this chapter concentrates on the usage of the SCRATCH 2 Offline Editor instead.

This chapter uses EDBOT V.5.2.0.1523 which allows **multiple robot** programming via three main software interfaces: SCRATCH, PYTHON and JavaScript, and with additional languages such as Node.js, Java and C# and other platforms such as Chromebooks, Macs and Linux (http://support.ed.bot/edbot-software-index.html). This version also accommodates the 17-servo configuration of the MINI and the functionality of MOTION-LIST/PAGE "0" and "-1" to make use of the EXIT MOTION-UNITs (http://emanual.robotis.com/docs/en/software/rplustask3/task_parameters/#motion-index-number).

In this chapter, an individual user is assumed, i.e. the user's PC is used as the local host (Port 8080) and the EDBOT tool is used in its server mode.

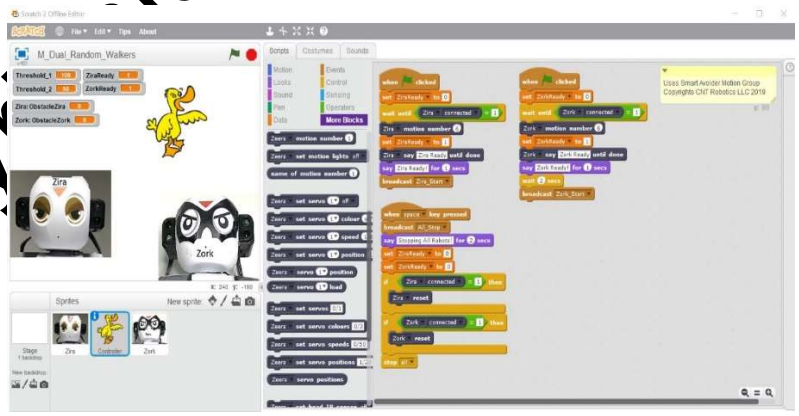

Fig. 8.1 Multi-robot application of the EDBOT tool with SCRATCH 2.

For this Chapter, the author assumes that the reader is already familiar with MIT SCRATCH 2 software. If not, the user is recommended to first read up on such works as Ford (2014), Warner (2015) or Vlieg (2016) to have a thorough understanding of the SCRATCH 2 language. In this Chapter, only selected SCRATCH 2 features, most applicable to robotics and unique to the workings of the Edbot software, would be presented.

8.1 Edbot's Hardware and Software Installation Requirements

The user will need to purchase the appropriate Edbot MINI kits (which has one IRSS-10 mounted on Port 1) and/or software product keys from Robots in School Ltd. (https://shop.ed.bot/collections/products). If the user has already obtained a MINI robot elsewhere, the user can just buy the Edbot product key from Robots in School Ltd. (https://shop.ed.bot/collections/products/products/edbot-software) or from ROBOTIS USA (http://www.robotis.us/edbot-software-product-key/). **In this book, the author assumes that the user is using the EDBOT tool (Version 5.2.0.1523 or higher).**

Hardware wise, the EDBOT software works with the ROBOTIS BT-210 (http://www.robotis.us/bt-210/).

Please view Video 8.1 for the typical steps needed to install the EDBOT software on the user's PC. **Importantly, each software product key will be bound permanently to the specific BT-210s used, thus the user will need to plan out carefully his or her "robots" organization if the user wants to use the EDBOT software with multiple robots.** Fig. 8.2 shows the author's EDBOT set up for 5 robots: Zeeb is a DREAM robot, Zomby is a PLAY700 robot, while Zeera, Zira and Zork are MINI robots.

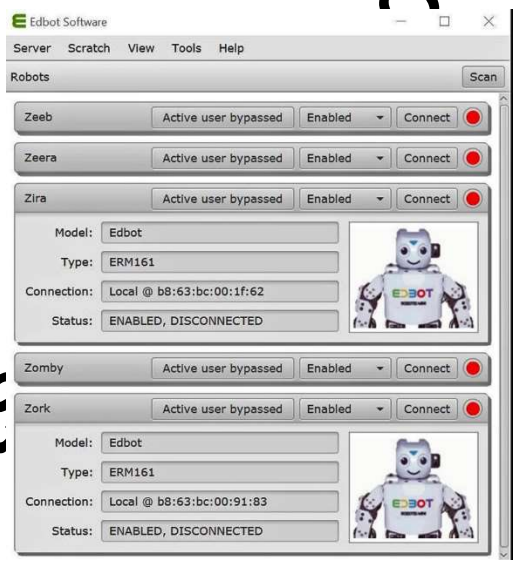

Fig. 8.2 Author's EDBOT Setup for his 2 MINI robots (Zira and Zork).

8.2 SCRATCH 2 Project File Conversion

The author used 2 MINI robots named "Zira" and "Zork" to create the SCRATCH 2 example codes presented in this Chapter, but the reader may use a different name for his/her robot(s), and thus will need to do an updating process for the SCRATCH SB2 files provided by the author.

8.3 SCRATCH 2 Blocks Provided with EDBOT Software

For the MINI, EDBOT provides 20 Stack Blocks and 7 Reporter Blocks which can be accessed under the "More Blocks" item of the SCRIPTS tab (see Fig. 8.8):

- Group 1 (see Fig. 8.9) includes Stack Blocks that work specifically with the MINI Default Motion Group so they are not used in this Chapter as we use Custom Motion Groups.
- Group 2 (see Fig. 8.10) are for general use, and the Motion Control blocks use the standard Motion Index Numbers as argument.

The EDBOT website has a support page at http://support.ed.bot/EDBOT-scratch2.html where the various Blocks are classified as: MOTION CONTROL, SINGLE SERVO, MULTI SERVO, SENSOR, SPEECH and GENERAL.

Fig. 8.8 EDBOT provides 20 Stack Blocks and 7 Reporter Blocks.

Fig. 8.8 also shows the standard SCRATCH Blocks which are grouped under different categories: **Motion, Looks, Sound, Pen, Data, Events, Control, Sensing and Operators**.

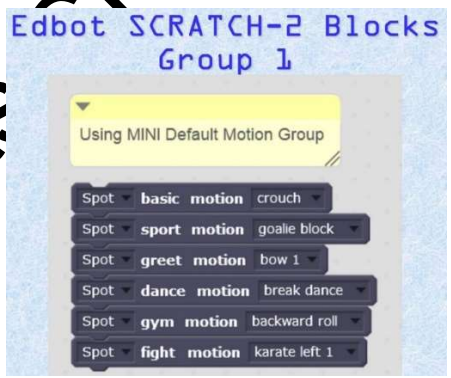

Fig. 8.9 Group 1 of EDBOT's Stack Blocks.

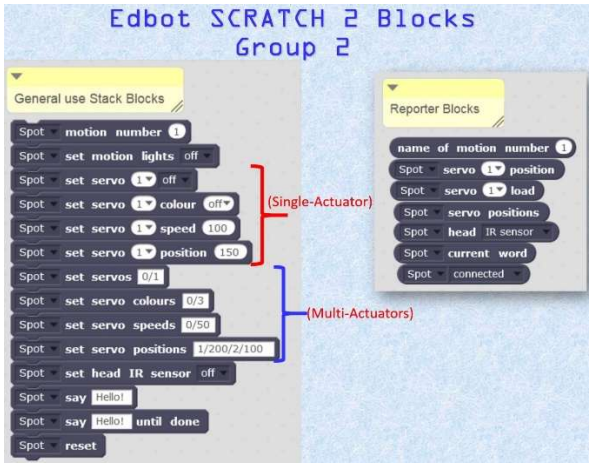

Fig. 8.10 Group 2 of EDBOT's Stack Blocks & Reporter Blocks.

8.3.1 MOTION CONTROL Blocks

Fig. 8.11 shows the currently supported EDBOT MOTION-related blocks:

- The "Name of Motion Number" Reporter Block is a unique EDBOT feature.
- The "Set Motion Lights" Stack Block is equivalent to the ROBOTIS "Joint LED Auto ON (Motion Playback)" command.

Fig. 8.11 EDBOT's Motion Control Blocks.

8.3.2 SINGLE & MULTI SERVO Blocks

Fig. 8.12 shows the currently supported EDBOT SERVO-related blocks:

- The "Single-Servo" Blocks correspond the various ROBOTIS commands such "Goal Position", "Goal Velocity" etc… for individual servos, and they function in a similar manner but with slower data throughput.

8.3.3 SENSOR Blocks

Sensor-wise, EDBOT currently assumes that there is only ONE IRSS-10 sensor connected to Port 1 of the MINI (see Fig. 8.13) – for its SCRATCH implementation. To access the full range of sensors and to use all four I/O Ports like in TASK previously, the user needs to switch to using PYTHON with EDBOT (see Section 8.7 and later).

Fig. 8.13 EDBOT's Single NIR Sensor Blocks.

8.3.4 SPEECH Blocks

Fig. 8.14 shows the Speech Blocks implemented by EDBOT. These "audio" SAY Blocks are different from the standard SCRATCH's SAY Blocks which are "visual display" and "non-audio" outputs.

Fig. 8.14 EDBOT's Speech Blocks.

8.3.5 GENERAL Blocks

These GENERAL or SYSTEM blocks relate to the overall operation of the OpenCM9.04-C Controller: "Reset" is a Stack Block and "Status" is a Reporter Block (see Fig. 8.15). "Reset" will turn off all hardware components and stop all operations on the OpenCM9.04-C, but the SCRATCH code will still be active. The "Status" block is handy to check if your MINI robot is "Enabled" or "Connected" to the EDBOT software on the Desktop Computer or not, especially if Bluetooth happens to take long time to connect properly when a SCRATCH project starts to run.

8.4 RC Walker with Independent Servo SCRATCH Project

In this section 8.4, the author's goal is to create a SCRATCH Project that is equivalent "as much as possible" to the "M_RC_Walker_IS.tsk3" (see Section 6.4), whereas Servos #1 (Left Shoulder) and #17 (Head) can be independently controlled from the effects of the currently performing MOTION-LIST/PAGE.

8.4.1 Differences between TASK & SCRATCH

Let's first discuss important differences between ROBOTIS TASK and SCRATCH+EDBOT:

- The RC-100 provides a 16-bit message that can tell the programmer about the status of all Buttons U-D-L-R-1-2-3-4-5-6 (see Section 6.4.1), while SCRATCH can access most keys of the standard computer keyboard but only as a "single" key being pressed. Fig. 8.16 shows the differences in handling "Button" or "Key" events between TASK and SCRATCH and it is obviously much simpler when using SCRATCH.

Fig. 8.16 Event Handling Differences between TASK and SCRATCH+EDBOT.

8.4.2 Keyboard Solution "M_RC_Walker16_IS.sb2"

This SCRATCH project uses the **Default Motion Group for the standard MINI with only 16 servos** and we'll use the following MOTION-LIST/PAGEs:

- MOTION-LIST/PAGE (1) is used for "Initial Position".
- MOTION-LIST/PAGE (19) is used for "Advance".
- MOTION-LIST/PAGE (20) is used for "Reverse".
- MOTION-LIST/PAGE (15) is used for "Right Turn".
- MOTION-LIST/PAGE (16) is used for "Left Turn".

Fig. 8.17 shows the Costumes used for robot Zira: "Idle" and "Ready".

Fig. 8.17 "Costumes" used for robot Zira.

Fig. 8.18 shows the MAIN SCRIPT for "M_RC_Walker16_IS:

- Block 1 is one of the "Event" type of Blocks and it would start whenever the user clicks on the FLAG Menu Item found on the Top Right corner of the STAGE Area.
- Blocks 2 and 3 belong to the "Looks" category. Block 2 displays the costume "Zira_Idle_m" onto the STAGE. Block 3 displays a Text Balloon, also on the STAGE and usually located at the top right of the associated SPRITE Zira, which says "Zira Not Ready!" for 2 seconds then this Text Balloon would disappear.

8.5 Dual RC Walkers SCRATCH Project

Although EDBOT can support multiple robot connections, the results from Section 8.4.2 show us that there is no feasible way to control two MINIs independently from each other as SCRATCH can only detect a "single" keystroke at a time from the operator. But it should be feasible to **control two MINIs synchronously** to do the same motions as commanded by the operator's keystroke sequence.

In this "Dual RC Walkers" project, we'll combine keystroke events with MESSAGE BROADCAST techniques to synchronously control two MINIs. We'll also use 3 SPRITEs for this project: "Duck" Controller and the two robots Zira and Zork (see Fig. 8.20). The "Duck" Controller will have a supervisory role of the connections between EDBOT+SCRATCH and the robots. It also intercepts the incoming operator's keystrokes and broadcasts the appropriate messages to the 2 robots for them to perform the commanded motions.

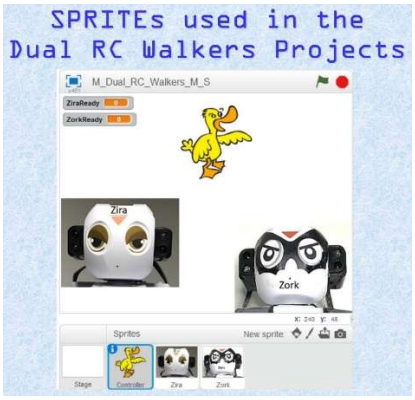

Fig. 8.20 Various "Keyboard Input" SCRIPTS in "M_RC_Walker16_IS.sb2".

Two versions will be described: a Simplified Version having a bare-bone structure just enough to get the job done, and a Verbose Version that can help us see the interactions between keystrokes, messages and robot behaviors.

8.5.1 Simplified Version "M_Dual_RC_Walkers_M_S.sb2"

The project "M_Dual_RC_Walkers_M_S.sb2" uses the **MOTION GROUP named Smart Avoider** from the file "MINI_Custom.mtn3".

The "Duck" Controller uses 3 types of SCRIPTs which are described in Figs. 8.21, 8.22 and 8.23.

Fig. 8.21 describes the Controller's FLAG Scripts, one for Zira and one for Zork (i.e. they are set **to run independently and simultaneously** when the operator clicks on the GREEN FLAG Icon). The reader can see that Zira and Zork share the same programming logic except for the use of different robot names in EDBOT-related Blocks. Thus, the author would use Zira's Script as demo code:

8.5.2 Verbose Version "M_Dual_RC_Walkers_M_V.sb2"

The "Verbose" version "M_Dual_RC_Walkers_M_V.sb2" is designed with several goals:

- Implementation of SPRITE-specific parameters "ZiraInMotion" and "ZorkInMotion" to obtain similar functionality to the TASK's MOTION-STATUS flags.
- All robots Sprite Scripts are now MESSAGE-driven, while all FLAG Scripts are reserved for the Controller Sprite for a more hierarchical but also distributed control structure.
- New messaging tasks are used to show details of the progress of keystrokes and broadcast messages during runtime.

Fig. 8.28 shows the FLAG Scripts used by the Controller Sprite to initialize Zira and Zork (let's use Zira's Script for demo):

Fig. 8.28 Controller's FLAG SCRIPTS for Zira and Zork in "M_Dual_RC_Walkers_M_V.sb2".

- First, SCRATCH sets Parameter "ZiraReady" to "0" and waits for the BT connection to Zira to get established.
- Next, EDBOT sets Zira's MOTION LED setting to OFF and voices out "Zira Connected".
- SCRATCH then displays the Text Balloon "Zira Connected!" for 2 seconds.
- Next, SCRATCH broadcasts "Zira_Connected" and waits for 3 seconds (so that Zira can execute its "Zira_Connected" MESSAGE Script (Fig. 8.29).
- Lastly, SCRATCH broadcasts "Zira_Start".

The other Scripts for Controller Sprite remain un-changed, thus they are not repeated here.

Fig. 8.29 displays "Zira_Connected" Script showing Zira's steps after receiving the message "Zira_Connected":

Fig. 8.29 Zira's "Zira_Connected" SCRIPT in "M_Dual_RC_Walkers_M_V.sb2".

8.6 Dual Random Walkers SCRATCH Project

In this Section 8.6, Zira and Zork are programmed as autonomous robots using the overall algorithm developed for Project "M_RandomWalker.tsk3" described in Section 6.6. Same as for the other Dual Robots Projects, 3 SPRITES (Controller, Zira and Zork) are used for this Project "M_Dual_RandomWalkers.sb2" which also uses the Smart Avoider Motion Group.

Fig. 8.32 shows the Controller's FLAG Scripts used separately for Zira and Zork to program their "Getting Ready" behaviors. Again, the author only describes in details Zira's behaviors as a demonstration for Zork's identical behavior:

Fig. 8.32 Controller's FLAG SCRIPTS for Zira and Zork in "M_Dual_Random_Walkers.sb2".

8.7 Using PYTHON

When using PYTHON with EDBOT, the user can get "under the hood" to learn more details about the interface between EDBOT and the ROBOTIS firmware used on the OpenCM9.04-C. The user also can get to other utilities that were not implemented in the SCRATCH API such as:

- Full access to all 4 I/O ports (i.e. we can use all 3 NIR Sensors IRSS-10 and DMS-80).
- Custom read/write to any address defined in the Control Table of the OpenCM9.04-C (i.e. we can now apply JOINT OFFSETs).

A PYTHON program will require more steps than a TASK or SCRATCH program to get the MINI to do a typical action, but its execution speed is much higher and it will allow the user to tap into the vast resources of other Python packages available at the Python Software Foundation (https://pypi.org/ and https://docs.python.org/3/contents.html). The web site Python Central is also a great resource for learning and practicing Python programming (https://www.pythoncentral.io/).

EDBOT V.5.2 was a major upgrade which changed fundamentally its Python API for the MINI, thus all sample codes discussed in this section used the EDBOT V.5.2.0.1523. The reader is assumed to have some basic skills in Python Programming, if not the author is recommending these books to get the reader better prepared for materials presented in this section (web resources abound for Python also):

- "Python for Tweens and Teens: Learn Computational And Algorithmic Thinking" (Bouras and Ainarozidou, 2017).
- "Python Crash Course", 2nd Edition (Matthes, 2019).

For more in-depth treatments of Python, the user is recommended to refer to these works:

- "Programming in Python 3", 2nd Edition (Summerfield, 2010).
- "The Python 3 Standard Library by Example", 1st Edition (Hellmann, 2017).

The author is using Thonny V.3.2.3 as the Integrated Development Environment (IDE) which is available for download at https://thonny.org/ , where it has a video demonstrating the basic usage of this IDE. Thonny V.3.2.3 also installs Python 3.7.5 for the user as part of its own installation and Thonny has its own GitHub page at https://github.com/thonny/thonny.

8.7.1 Basic Steps for PYTHON programming with EDBOT

The complete EDBOT's PYTHON API is available at this link http://support.ed.bot/edbot-python3.html which has information on how to set up and use EDBOT with PYTHON. The author assumes that the reader has installed EDBOT and configured it properly for the user's robot(s) – please review Section 8.1 and Video 8.1 as needed. The author uses THONNY 3.2.4 as his Integrated Development Environment, and it is bundled with PYTHON V. 3.7.5.

Let's first assume that the user has started EDBOT and connected successfully to a chosen robot such as Zira (see Fig. 8.44).

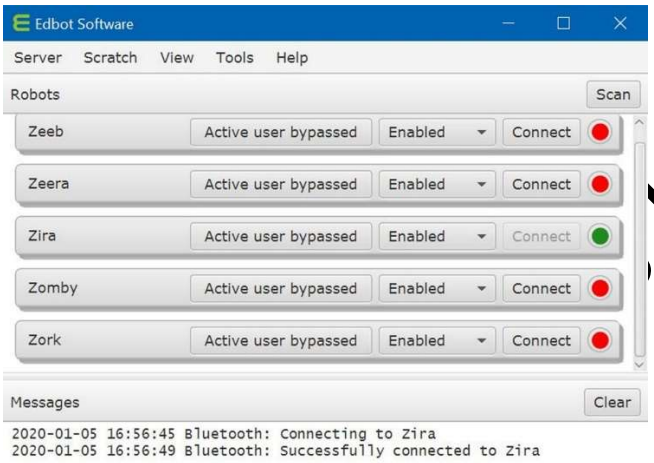

Fig. 8.44 EDBOT connected to robot Zira (author's setup).

Next, there are definite and orderly steps that the user must perform in order to successfully control one or multiple MINI robots in a PYTHON program:

1. Import the "edbot" package from the Python Package Index (PyPI) web site (https://pypi.org/project/edbot/), and also "time" and "sys" as needed:
   - import edbot
   - import time *# if time functions are used*
   - import sys *# as we'll need to use sys.exit()*
2. Connect to EDBOT as an Edbot Client via the local Port 8080 on your personal computer with the following Python statements:
   - ec = edbot.EdbotClient("localhost", 8080)
   - ec.connect()
3. Create a variable (e.g. "robot1") and assign to it the user's robot name (e.g. "Zira") as previously configured in EDBOT (see Figs. 8.2 and 8.44):
   - robot1 = "Zira"
4. Check the previous robot name with EDBOT and verify its configuration, i.e. whether the user's robot is an "edbot", "dream" or "play" robot. The MINI is an "edbot" type and if there is a configuration error, Thonny ought to do a "system exit":
   - if not robot1 in ec.get_robot_names("edbot"):
   - print(robot1 + " is not configured")
   - sys.exit()
5. Wait for actual connection between Thonny (via EDBOT) and the physical robot:

- print("Waiting to control " + robot1 + "... ", end="")
- ec.wait_until_active(robot1)
- print("Got control of " + robot1 + "!")

8.7.2 Sensors Throughput Applications

Let's get started with a basic Python program reading and displaying the current readings of two IRSS-10 sensors out of Ports 1 and 4, a DMS-80 out of Port 2, and the Present Positions for Servos #1 and #7 for robot "Zira".

But before we can get into those gory details, we need to examine the Data Structures used by EDBOT to keep track of the real-time values of **all its sensors and actuators connected to each robot** configured in EDBOT's setup.

Each robot's sensors-data structure is a PYTHON dictionary named "reporters" (see link port.ed.bot/edbot-python3.html#reporters), which is a list of paired "keys" and "numerical values" shown below as an example for robot "Zira":

```
"reporters": {
  "port1": 10,
  "port2": 650,
  "port3": 335,
  "port4": 22,
}
```

For example, the current raw value for the "left" IRSS-10 sensor (with "port1" key) is "10" and the current raw value for the "port2" DMS-80 sensor is valued at "650". "So far, so good", the reader must reckon at this point! But it is getting bad quickly as the next paragraph will show.

In order to get to this specific Zira "reporters" dictionary from inside a Python program, we need to use a method named ***"ec.get_data()"*** (https://support.ed.bot/edbot-python3.html#_get_data) which will return a "huge" dictionary containing sensors data (+ other data structures), i.e. dictionary within dictionary:

```
{
 "server": {                    # server information
   "version": "5.0.6.1247",
   "platform": "Windows 10, 10.0.17134.523, amd64"
 },
 "auth": "9TBvXvf9",            # private session token
 "initComplete": True,          # true after connect() returns
 "robots": {
   "Zira": {                    # name of the robot
     "enabled": True,           # enabled?
     "connected": True,         # Bluetooth connected?
```

```
    "reporters": {
      ...
    },
    "activeUser": "Python...",   # currently active user
    "model": {                   # the robot model
      "name": "Edbot",
      "type": "ERM162",
      "key": "edbot"
    }
  }
 },
```

Unfortunately, the Zira's "reporters" dictionary, that we are interested in, is "buried" inside the "robots" dictionary highlighted in yellow as shown above. This means that we need to use the value "Zira" and apply it to the "robots" dictionary in order to extract out the "reporters" dictionary of interest, which is then can be extracted further out for its individual "sensor" key and its corresponding numerical value! The implementation of this complex procedure will be shown later in Fig. 8.4.8.

8.8 MINI PYTHON/EDBOT Applications

Using PYTHON with EDBOT will give us the opportunities to compare PYTHON solutions to similar TASK solutions created for the same type of projects such as "RC Walker" and "Random Walker", but it will also allow us to extend the capabilities of the MINI with "Dual MINIs" and "Web Cam" projects using Threading and OpenCV libraries (features that are now not available using ROBOTIS TASK V.3).

Let's first review the key design features and constraints used in EDBOT:

1. In the TASK project "M_RC_Walker_IS.tsk3" (see Section 6.4.3), the parameter "MOTION-STATUS" was key to the logic design of this TASK program, but there is no such equivalent parameter in EDBOT.
2. Previously, in SCRATCH/EDBOT applications (see Section 8.4.2), the Play-Motion Block would wait until the called Motion is finished playing before letting the next Block activate. However, when PYTHON is used with EDBOT, the method ***run_motion()*** can be used with either WAIT or NO-WAIT option (http://support.ed.bot/edbot-python3.html#run_motion).
3. Also, in previous SCRATCH/EDBOT applications, there was no way to set the JOINT-OFFSET of the XL-320 servos. But the PYTHON API has a method called ***set_custom()*** that can now be used to set these JOINT-OFFSET parameters for Servos #1 and #17.
4. The EDBOT PYTHON API does not allow access to sensors and servos data during a MOTION PLAY.

We'll use the Pygame library for the display and keyboard interfaces (https://www.pygame.org/docs/ref/display.html and https://www.pygame.org/docs/ref/key.html) and it turns out that Pygame also implements the same "sequential" and "single-key" input functionalities that SCRATCH was using. Thus, the combined-buttons approach of TASK programs cannot be reproduced in PYTHON/EDBOT applications also.

8.8.1 RC Walker with Independent Servo Control

In this project "M_RC_Walker_IS.py", the Pygame package is used to monitor the user's keyboard inputs and to provide graphical screen outputs at run time. The web site "https://www.pygame.org/docs/" has all the official PyGame documents that the reader can consult as needed. If the reader prefers a regular textbook, the author recommends the work by Kinsley & McGugan (2015).

The MINI robot named Zira is used in this project and Fig. 8.51 is a typical screen capture at run time:

Fig. 8.51 Screen capture at run time for "M_RC_Walker_IS.py".

8.8.2 Dual RC Walkers

The reader may recall the "Dual RC Walkers" project using SCRATCH (see Section 8.5) where the two MINIs' motions "had to be synchronized" due to the constraint on SCRATCH that can handle only one "key-stroke" at any one time. As PYTHON will allow EDBOT programming at a lower level, the author's hope was that independent Remote Control of two MINIs could be achieved when using PYTHON.

Thus, for this project "M_Dual_RC_Walkers.py", the goal is to independently "remote-control" two MINIs (Zira and Zork) from the same PYTHON program. The usual arrow keys U-D-L-R will be used to control Zira (as "robot1") and the keys W-Z-A-S will be the equivalent "arrow" keys for controlling Zork (as "robot2") – see Fig. 8.61.

"M_Dual_RC_Walkers.py"
Variables Definition

```
30 robot1 = "Zira"
31 robot2 = "Zork"
32 robot1_mode = 'n' # f for Forward, b for Backward,
33 robot2_mode = 'n' # f for Forward, b for Backward,
34 background_image_filename = 'Zira_Zork_Ready.jpg'
35 SCREEN_SIZE = (927, 349)
36 message1="1) Tap Arrow Keys to move Zira"
37 message2="2) Tap [w,z,a,s] Keys to move Zork"
```

Fig. 8.61 Variables Initializations in "M_Dual_RC_Walkers.py".

Fig. 8.62 lists the user methods defined for this project, they are fewer than for the previous project "M_RC_Walker_IS.py" because the author wanted to show a different way to implement Motion Control for the MINIs whereas the EDBOT method ***run_motion()*** is used directly in the Main Method:

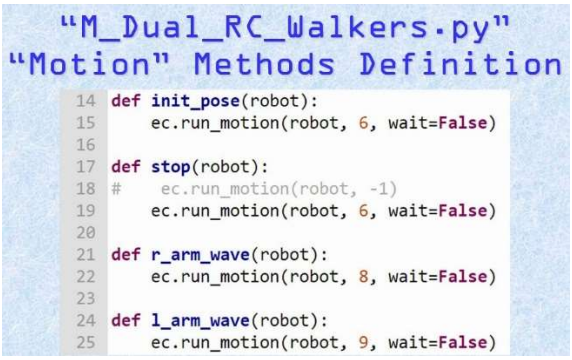

Fig. 8.62 User Methods in "M_Dual_RC_Walkers.py".

- The only "new" user method is ***stop(robot)*** whereas the author wanted to use MOTION (-1) (Line 18) as applied in previous TASK and SCRATCH projects, unfortunately the EDBOT V. 5.2.0.1523 does not yet support this feature in PYTHON (even though it supported this feature in SCRATCH – see Fig. 8.26). This issue will probably get sorted out in a later version of EDBOT, thus the author left Line 18 in the source code for the reader to try out in the future.
- So for now, method ***stop()*** corresponds simply to run the MOTION (6) to get the robot into Initial Pose.

8.8.3 Autonomous Random Walker & Obstacle Avoider

The reader may recall from the Random Walker SCRATCH project (Section 8.6) that SCRATCH would only support the use of **one** IRSS-10 sensor connected to **Port 1**, thus this SCRATCH solution was more limited in its response when an obstacle was found. On the other hand, the PYTHON API allows the use of all sensors on all 4 Ports. Thus, the PYTHON project "M_RandomWalker.py" can use two NIR sensors to respond to a found obstacle in the same way as the TASK-equivalent project "M_RandomWalker.tsk3" in Section 6.6.

The TASK project "M_RandomWalker.tsk3" relied heavily on using MOTION (0) and MOTION (-1), as well as the CALLBACK Function to provide the most recent Sensors Data at any point in this TASK program. Unfortunately, these two features are not available currently in EDBOT/PYTHON V. 5.2.0.1523:

- MOTIONs "0" and "-1" do not work properly in PYTHON, although they worked fine for SCRATCH 2.
- By design, EDBOT does not update sensor and servo data while a MOTION-LIST/PAGE is playing.

Thus, although the overall goals and algorithm are the same between the TASK and PYTHON versions for the Random Walker Project, the reader will see that some implementation details had to be changed in the PYTHON version.

Fig. 8.63 lists the Definition of Variables used in "M_RandomWalker.py":

Although both robots "Zira" and "Zork" have their respective Variables defined for them (Lines 160-161 and Lines 162-163) this PYTHON code is made to control only 1 robot, but it can be either Zira or Zork. Variables "robot1", "robot2", "obstacle1", "obstacle2" and "threshold1" are Global.

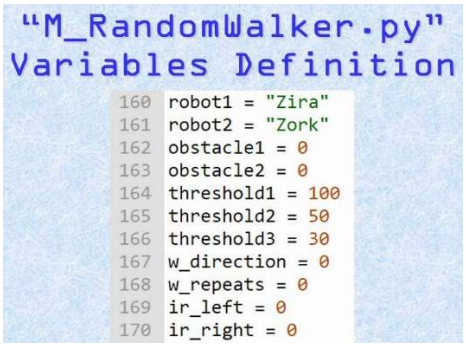

Fig. 8.68 Variables Definition in "M_RandomWalker.py".

- Please note that "obstacle1" and "obstacle2" are designed to be set to "0" when no obstacle is found, and when an obstacle is found by the respective robot ("robot1" or "robot2"), they will be set to "1".

There are many methods used in the program "M_RandomWalker.py":

- The various MOTION related methods already described in previous PYTHON projects, such as ***init_pose(), forward()*** and ***l_arm_raise()***, won't be repeated here.
- Two methods of robot stoppage are defined - ***reg_stop()*** and ***emer_stop()***. Method ***reg_stop()*** is to be used when the robot encounters an obstacle when it **has just concluded** a set of randomized moves. Method ***emer_stop()*** is to be used when the robot encounters an obstacle **while still performing** a set of randomized moves (see Fig. 8.69).
- Two "random-based" methods are used to set the robot's move direction and the number of repeats for such move – ***set_direction()*** and ***set_repeats()*** (see Fig. 8.70).
- Method ***walk_execute()*** takes the previous randomized move-direction and move-repeat values and commands the robot to perform such-directed moves (Figs. 8.73 and 8.74).
- Method ***set_obstacle()*** sets Parameter "obstacle" to either "obstacle1" or "obstacle2" depending on which robot among "robot1" or "robot2" is being tested (Fig. 8.71).
- Method ***check_NIR()*** is called by Method ***walk_execute()*** at the end each robot move to check if an obstacle is found, and it issues the start of an "emergency stop" when needed (Fig. 8.72).
- Method ***flash_LED()*** is called at the conclusion of a **successful** obstacle avoidance procedure (Fig. 8.75).

8.8.4 Dual Random Walkers with Thread Programming

This project was described in Section 8.6 for its SCRATCH implementation which was using the concepts of Event Programming via "Message Broadcast" and "Message Received" between 3 independent Sprites (Zira, Zork and Duck as the Game Controller). The same Event Programming approach will be used in this PYTHON project using PYTHON-equivalent constructs which are "Thread" and "Event" objects from the "**threading**" module of the PYTHON Standard Library (https://docs.python.org/3/library/threading.html#). The following web page was very helpful to the author by providing working code snippets used in this project: https://pymotw.com/3/threading/index.html#module-threading – it is maintained by Doug Hellmann.

This PYTHON project "M_Dual_RandomWalkers.py" uses the same approaches as previous projects regarding EDBOT robot control and PYGAME keyboard/graphics interfacing so only the newer concepts of

threading and event signaling will be described further in the following paragraphs. If the reader is interested in textbooks to learn more about "threads" or "concurrency" in general, the author would suggest Hellmann (2017) and Nguyen (2018).

The author was delighted to find that all Methods created for the Single Random Walker project (with Zira) in Section 8.8.3 were re-usable in the Dual Random Walkers project (with Zira and Zork). The Main Endless Loop used in the Single Random Walker project needed to be recast as a Thread for Zira and duplicated as another Thread for Zork in the Dual Random Walkers project, with minor code changes to adjust for the robot names, and some code additions at the beginning and at the end of each of these Threads. A third new Thread had to be created to act as the Game Closer, while the Main Method acts as the Game Starter.

Fig. 8.81 shows all the import modules used in "M_Dual_RandomWalkers.py".

"M_Dual_RandomWalkers.py"
Import Packages

```
8  import sys
9  import time
10 import edbot
11 import random
12 import pygame
13 from pygame.locals import *
14 import threading
```

Fig. 8.81 Modules imported in "M_Dual_RandomWalkers.py".

In PYTHON, a thread can be simply created as a regular function like the ones previously created for various robot maneuvers (https://en.wikibooks.org/wiki/Python_Programming/Threading). As we plan to use the Event Object formalism (https://docs.python.org/3/library/threading.html#event-objects), each thread will also need a special command called ***message.wait()*** which indefinitely blocks the execution of **all codes that follow** the ***message.wait()*** command. When the ***message.set()*** command is asserted in the main or in other threads, then this thread's code execution restarts and proceeds as normal for the rest of the thread/method. Thus the ***message.wait()*** command is the equivalent of the SCRATCH's HAT command "**When I receive Message**".

As an example, let's look at the details of Thread ***go_Zira()*** (see Fig. 8.82):

- Line 164 lists the Global Variables used for this thread. "game_start" and "game_over" are Event objects, while reset, winner, robot1, obstacle1, threshold1, threshold2, threshold3) are standard Variables defined in the Main Method.
- Line 166 uses the command ***game_start.wait()*** which effectively blocks the execution of the remaining code for this thread which then can only wait for a signal ***game_start.set()*** from another thread/method (which will be the Main Method for this case – see Fig. 8.86). Thus the signal ***game_start.set()*** is equivalent to the SCRATCH's **Broadcast Message** block.
- Once the signal ***game_start.set()*** is received by this thread, Line 167 prints out, on THONNY's output console, the message "Thread go_Zira starts".

8.8.5 Color Tracker with Web Cam using OpenCV

The reader may recall the TASK project "M_CameraTracker.tsk3" in Section 7.6, leveraging the R+m.PLAY700 App to use the camera from a smartphone. This project was limited because of the R+m.PLAY700 App itself and because of the weight of the smartphone as the MINI could only turn left or right in a stable manner.

But now that we are using PYTHON from a Desktop Computer, inexpensive and light-weight USB webcams are available to be mounted of the MINI's head. The author uses a Logitech C270 with its stand removed for Zira and Zork (see Fig. 8.89). Furthermore, USB cables can be up to 25 ft long, so the MINI should have good freedom of movement, although it does have to drag that USB cable around which may influence its walking moves.

Fig. 8.89 Zira and Zork equipped with webcams mounted on their head pieces.

Another advantage when using PYTHON which is a widely adopted language is the large repository of modules/tools found at the Python Software Foundation https://pypi.org/. This section 8.8.5 would illustrate the use of the OpenCV module and a related module called NumPy (https://pypi.org/project/opencv-python/). If the reader is using THONNY, it is recommended to use the "Manage Package" tool to search for the module "opencv-python" and install it. This process will install all related modules for the reader in one step. The author is using OpenCV-Python V.4.1.2.30 and NumPy V.1.18.1.

The field of Computer Vision or Machine Vision is large and complex, thus in this section the author is only trying to show that, with a bit of work, the reader can incorporate Machine Vision capabilities to a MINI robot. References for Machine Vision, OpenCV and NumPy abound, so the author is only recommending a few that he is familiar with:

- "Learning OpenCV 3" by Kaehler and Bradski (2017), this book is written for the C/C++ programming language but it is a thorough reference to keep.
- For Cookbook types of OpenCV book references in Python, the author has used "Learn OpenCV 4 by Building Projects" by Escrivá and Mendonça (2018) and "OpenCV 3 Computer Vision with Python Cookbook" by Spizhevoy and Rybnikov (2018) and had found them useful.
- Web resources (free and at cost) for OpenCV-Python and NumPy can be found at many places and here are a few:
  - https://opencv-python-tutroals.readthedocs.io/en/latest/index.html
  - https://www.learnopencv.com/about/
  - https://www.pyimagesearch.com/
  - http://www.numpy.org/ and https://docs.scipy.org/doc/numpy/.
  - http://cs231n.github.io/python-numpy-tutorial/

The author assumes that most readers are aware that a color digital image consists of physical pixels in a 2-dimensional array such as 1280 x 1024 or 3840 x 1600 (https://en.wikipedia.org/wiki/Color_image). Each of these physical pixels is in fact a triplet of pixels in the primary colors Red, Green, and Blue (also known as the RGB Color Model - https://en.wikipedia.org/wiki/RGB_color_model). The RGB Color Model is

needed to capture a color scene from the real world into a video camera sensor and to display the same color scene on a computer monitor (https://www.visiononline.org/userassets/aiauploads/file/cvp_the-fundamentals-of-camera-and-image-sensor-technology_jon-chouinard.pdf). When these triplet light sources from a color display reach the human eye, a very complex bio-physical process occurs to give us the sensation of color (https://www.olympus-lifescience.com/en/microscope-resource/primer/lightandcolor/humanvisionintro/). In turn, we usually describe those color sensations as red, green and blue (the primary colors), but also as yellow or brown or violet (generally called "hue"). For example, to create the "yellow" sensation we would have to turn on the "red" and "green" light sources while providing no "blue" light. Thus, from the point of view of color image processing, we would have to use at least two parameters (Red & Green) to define a single hue (Yellow). For this reason, other color models are created (derived from the RGB model) to more closely align with how humans classify perceived colors (https://en.wikipedia.org/wiki/HSL_and_HSV), where we would need only one parameter to describe Hue (with the other two parameters S or L/V used to describe more subtle color perceptions such as vividness or brightness of a particular color perceived). We will be using the HSV Color Model in the Python projects described in this Sections 8.8.5 and 8.8.6.

First let's look at the project "M_RC_Vision_HSV.py" where the user can remotely control the movement of Zork (therefore the camera – see Fig. 8.90) to position a central Region of Interest (ROI) onto an object of interest to display its HSV data. This project also gives the user the option of recording and pausing multiple video clips into a single video file.

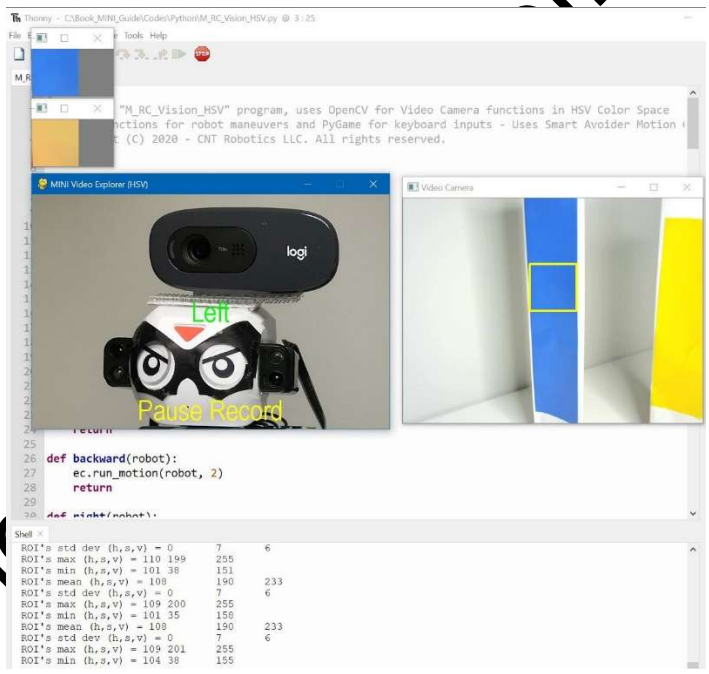

Fig. 8.90 Run-time Displays for "M_RC_Vision_HSV.py".

In this project "M_HSV_Tracker_Walker.py", the user can specify either "Blue" or "Yellow" objects for the robot Zork to autonomously search for and then, once found, approach it within a given distance controlled by the Left and Right IRSS-10 sensors on the MINI.

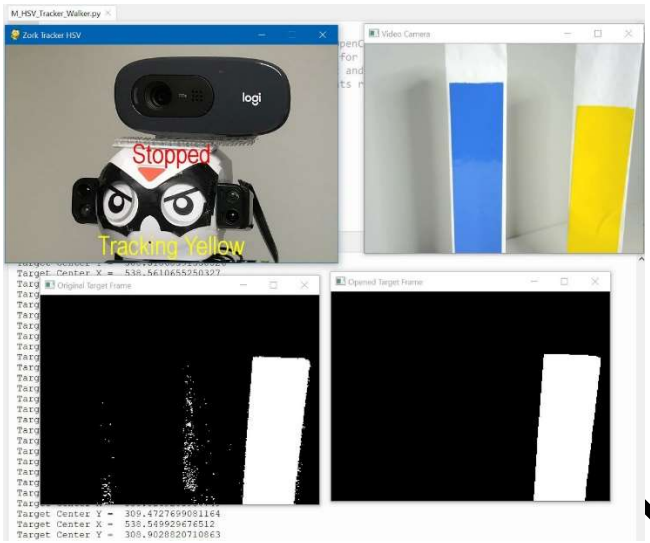

Fig. 8.102 "M_HSV_Tracker_Walker.py" can be used to track "Blue" or "Yellow" objects.

8.8.6 Dual Color Trackers with Web Cam using OpenCV & Threading

For this last PYTHON project, the code developed for "M_HSV_Tracker_Walker.py" is adapted into suitable Threads for Zira and Zork and the author chose a "game" orientation for this Thread Programming application project "M_Dual_HSV_Trackers.py".

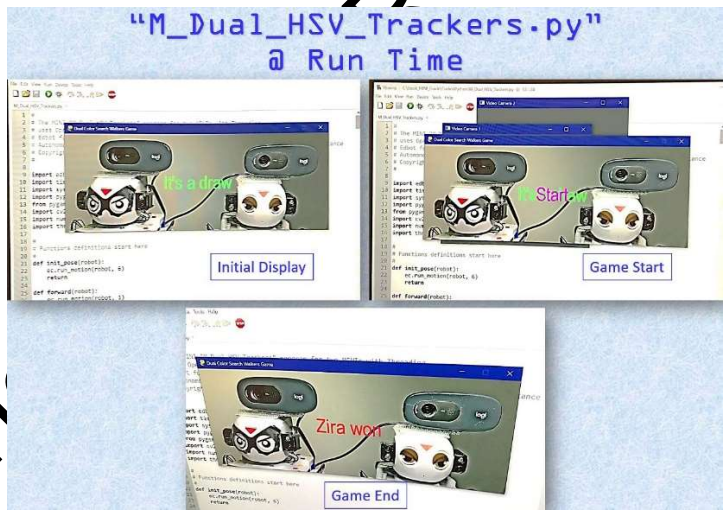

Fig. 8.128 Three Screen Captures during Video 8.13 for "M_Dual_HSV_Trackers.py".