

Projects Guide For ROBOTIS ENGINEER Volume 2

By Chi N. Thai



CNT Robotics LLC, Buford

2021

Chapter 4: Enhanced Pan-Tilt Commando with RPi4B

Fig. 4.1 shows the “Enhanced” PTC (E-PTC) used for projects described in this Chapter 4:

- An RPi4B, with 8 GB RAM and Pi Camera running on 64-bit Raspian, is used as a “Co-Controller” to the CM-550, replacing the RPi0W and the Windows PC used in the earlier chapters described in Thai (2020) (i.e. Volume 1). Although the RPi4B would do the main computing work for most projects in this chapter, within the ROBOTIS Dynamixel Network Paradigm the CM-550 retains the “Top-Controller” designation (see Section 1.4 of Volume 1). This RPi4B shares into the CM-550’s LiPo battery using a 12V-to-5V converter (see Appendix B for more details about setting up the RPi4B and CM-550, hardware-wise and software-wise).

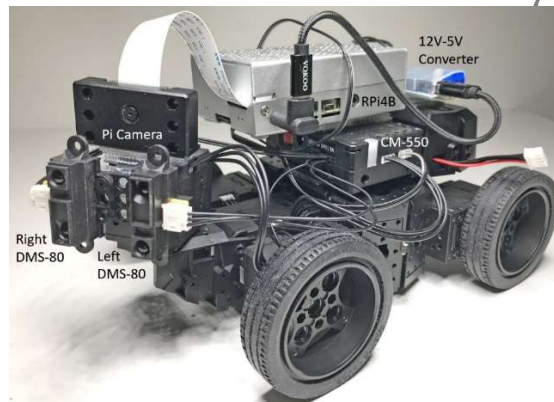


Fig. 4.1 “Enhanced” PTC with RPi4B and Pi Camera.

- The CM-550 is further equipped with two DMS-80 sensors via its GPIO Ports 1 and 2.
- The reader also needs to make sure that the TASK App is of V. 3.1.5 or above for the example codes in this Chapter to work properly.

This “E-PTC” robot was constructed to allow the exploration of several ideas/concepts:

1. Provide additional hardware and software capabilities to the CM-550 by using a tightly coupled general-purpose SBC such as the RPi4B via wired Serial Communications.
2. Introduce the concept of Independent Dynamixel Networks (IDN) which allows some actuators to be controlled directly by the RPi4B while other actuators remain under the control of the CM-550 (Sub-Section 4.5.1).
3. Develop a Supervisory Control (SC) role for the Desktop PC which can control multiple RPi4B+CM-550 types of robots via Bluetooth and/or WiFi Communications (Sub-Section 4.5.2).

Inasmuch as possible, Chapter 4 would maintain the “parallel tracks” approach used in Volume 1 by developing first the TASK and MicroPython tracks for the CM-550, followed by the Standard Python tracks designed for the RPi4B and Desktop PC, and lastly with the C/C++ tracks created for the RPi4B and Desktop PC.

The reader will see that the “glue” that keeps all the above component systems working together is the familiar Remocon Packet amply illustrated in Volume 1. Please see Fig. 4.2 for the overall communications scheme used for most of Chapter 4, except for Section 4.5 where the IDN concept is implemented via the Dynamixel SDK which adds the features of Dynamixel Packets (Protocol 2), and via the use of a new ROBOTIS hardware module called Dynamixel HAT (DXL-HAT), designed exclusively for the RPi series.

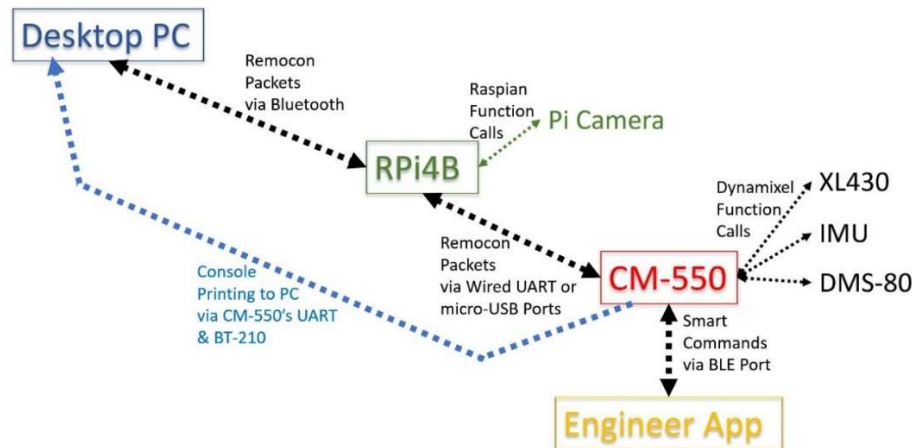


Fig. 4.2 Overall Communications Scheme used for Projects in Chapter 4.

Fig. 4.2 also hinted that Program Design complexity went up a notch, as 3 levels of programming would need to be coordinated from now on!

The TASK and MicroPython programs developed in this Chapter 4 (and in later chapters as well) would exhibit both autonomous and remote-controlled behaviors, for example:

- The human operator may issue Manual RC commands to the E-PTC robot from the RPi4B, the Desktop PC or a Mobile Device, but the CM-550 would use its DMS-80 sensors to perform its own Autonomous Obstacle Avoidance maneuvers when needed, thus overriding the human operator’s RC commands in such situations.
- The RPi4B may be in its autonomous Color Object Tracking mode via the Pi Camera and it is sending Object Tracking data over to the CM-550 for it to determine proper robot maneuvers to keep a Color Object centered within the Pi Camera’s viewport. However, if erroneous Object Tracking Data are being sent, resulting in the E-PTC robot potentially driving into the object itself, then the DMS-80s would kick in to prevent such an event to happen.

Section 4.1 is written for TASK coders and concentrates on the TASK programming requirements at the CM-550 level (see Fig. 4.2). If the reader is a MicroPython coder, please skip to Section 4.2 to read about the equivalent MicroPython programming requirements for the CM-550. Sections 4.3, 4.4 and 4.5 are written for readers who have gone through Section 4.1 or Section 4.2.

4.1 Using TASK

Two TASK projects are described in Section 4.1, and both used an OTG (or plain) USB cable to link the RPi4B to the CM-550's micro-USB Port, and a BT-210 to link the Desktop PC to the CM-550's UART Port (see Appendix B for more details on the hardware used and alternate options):

1. The first one establishes a Dual Control framework so that the CM-550 can be controlled by either the RPi4B and a Mobile Device, or both at the same time. This TASK solution is named "PTC_RCSD_VSR_PC_RPi.tsk3".
2. The second one adds an Autonomous Obstacle Avoidance feature to the previous Dual Control framework, and this TASK solution is named "PTC_SA_RCSD_VSR_PC_RPi.tsk3".

4.1.1 Dual Control from Mobile Device and/or PC Virtual Interface

The goal for this project to use a Mobile Device such as a Tablet or a Phone, via the Engineer App (see Fig. 4.3), to act as the main Remote Controller for a human operator to issue maneuvering commands to the E-PTC, via Touch Areas (i.e., via the CM-550's BLE Port).

Fig. 4.3 shows that there are three main groups of commands and two independent "modes" (SD and RPi/PC) that the human operator can set for the robot:

1. The "ROBOT" group has the usual motion-direction commands such as Forward/Backward/Left/Right.
2. The "CAMERA" group has the Tilt and Pan commands that can be issued to re-orient the Pi Camera.
3. The "Faster" and "Slower" Touch Areas can be used to change the SPEED of the robot on the fly.
4. The "SD" Touch Area in Fig 4.3 can be used to switch ON/OFF the "SD" mode:
 - a. When "SD" mode is ON (Yellow color), the ROBOT, CAMERA and SPEED commands can be issued by the operator.
 - b. When "SD" mode is OFF (Gray color), the ROBOT, CAMERA and SPEED commands are not available to the operator.
5. Similarly, the "RPi/PC" Touch Area in Fig 4.3 can be used to switch ON/OFF the flow of Remocon Data Packets coming from either the RPi or PC platforms.
6. It is expected that the two modes "SD" and "RPi/PC" can be switched ON/OFF separately or at the same time by the operator.

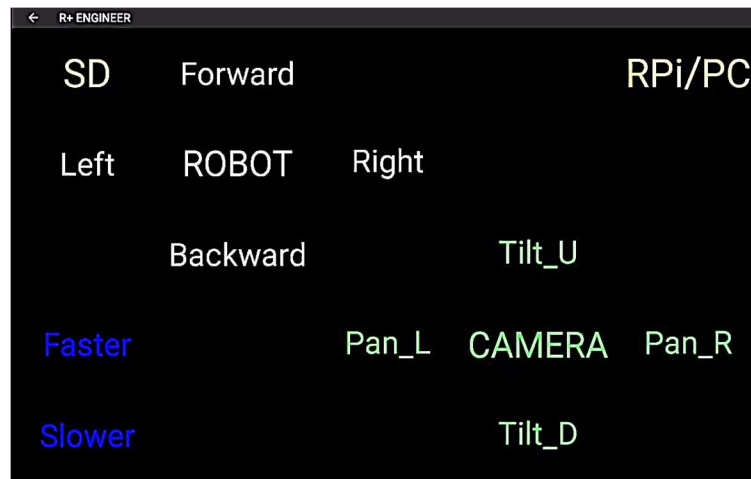


Fig. 4.3 Menu Screen used on the ENGINEER App for Dual Control Project.

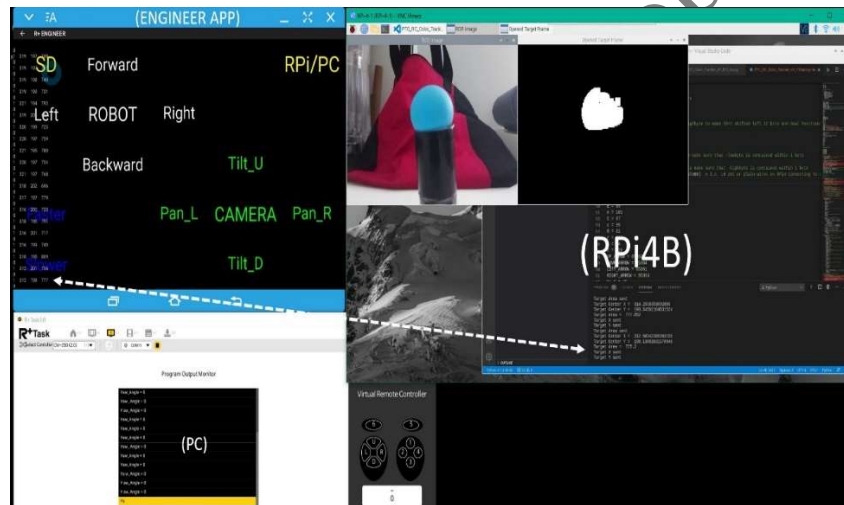


Fig. 4.4 Run-time Outputs from Configuration recommended by the author.

- The CM-550's Remote Port (Address 43) can receive and transmit Remocon packets from/to the PC or RPi4B depending on how the user sets it up. Line 342 showed that the author used the micro-USB COM Port to connect the CM-550 to the RPi4B via a USB cable.
- The CM-550's App Port (Address 36) can receive and transmit SMART commands to the ENGINEER App running on a Mobile Device. The CM-550's BLE COM Port is best used for this function thus Line 344 showed that the App Port is set to work through the BLE COM Port.
- The CM-550's Task Print Port (Address 35) is the Destination Port for the various PRINT and PRINT-LINE commands issued from the programmer's TASK code. Line

346 showed that the author used the CM-550's UART COM Port to connect the CM-550 to the Desktop PC via a BT-210 Receiver.

Ideas for further explorations (IFFE 4.1):

4.1.1.a: During runtime for the program "PTC_RCSD_VSR_PC_RPi.tsk3", the reader may have noticed that, at times, the SD and RPi_PC modes were not easy to turn ON/OFF via the Mobile Display's Touch Areas, this was because the operator's finger may have lingered in these Touch Areas "too long". One possible solution is to use a WAIT WHILE Loop that terminates only when a Finger Release is detected (after the detection of the original finger press, of course). The example program "PTC_RCSD_VSR_PC_RPi_WR.tsk3" shows these two WAIT WHILE Loops on Lines 38 and 52.

4.1.1.b: (applicable after Section 4.3 or 4.4) When both SD and RPi_PC modes are ON during runtime for the program "PTC_RCSD_VSR_PC_RPi.tsk3", the reader may have noticed that the robot had a repetitive "stop-and-track" behavior when it was trying to track the user's color object. This was a "side-effect" of the currently programmed action for the robot to do, when no Touch Area had been detected during a cycle of the Main Endless Loop, the robot is to STOP (see Line 109). The new code "PTC_RCSD_VSR_PC_RPi_NS.tsk3" provides a more "continuous" tracking behavior (see Lines 113-116). Which behavior did the reader prefer? The "continuous-tracking" behavior seemed less "intelligent" to the author as it took a longer time to settle its tracking behavior, upon a color object standing still, as compared for the "stop-and-track" behavior, this was "counter-intuitive"! Why so?

4.1.2 Dual Control with Autonomous Obstacle Avoidance

This section describes an "added" feature to the Dual Control capability demonstrated in the previous Section 4.1.1. Fig. 4.1 describes how the Left and Right DMS-80s are mounted in front of the E-PTC robot, and in this project, the Pan-Tilt servos will stay at their initial Goal Positions during run time. Therefore, with this configuration, the E-PTC robot can detect a "single" obstacle only in 3 ways: right in front of it, or to the left or to the right of it.

4.2 Using MicroPython

This section is written for MicroPython coders who did not read Section 4.1 (but if the reader did study Section 4.1 before getting here, please bear with some "unavoidable" information repetitions)

In early 2021, the ROBOTIS MicroPython API is published at this link <https://emanual.robotis.com/docs/en/edu/pycm>, but it is not quite complete yet at present (Summer 2021): for example, the SMART Functions are not yet documented.

Two MicroPython projects are described in this Section 4.2 and both used a USB cable to link the RPi4B to the CM-550's micro-USB Port, and a BT-210 to link the Desktop PC to the CM-550's UART Port (see Appendix B for more details on the hardware used and alternate options):

1. The first one establishes a Dual Control framework so that the CM-550 can be controlled by either the RPi4B or a Mobile Device, or both at the same time. This TASK solution is named “PTC_RCSD_VSR_PC_RPi.py”.
2. The second one adds an Autonomous Obstacle Avoidance feature to the previous Dual Control framework, and this solution is named “PTC_SA_RCSD_VSR_PC_RPi.py”.

4.2.1 Dual Control from Mobile Device and/or PC Virtual Interface

The goal for this project to use the Engineer App running on a Tablet or a Phone (see Fig. 4.17) to act as the main Remote Controller for a human operator to send maneuvering commands to the E-PTC, via Touch Areas and through the CM-550’s BLE Port.

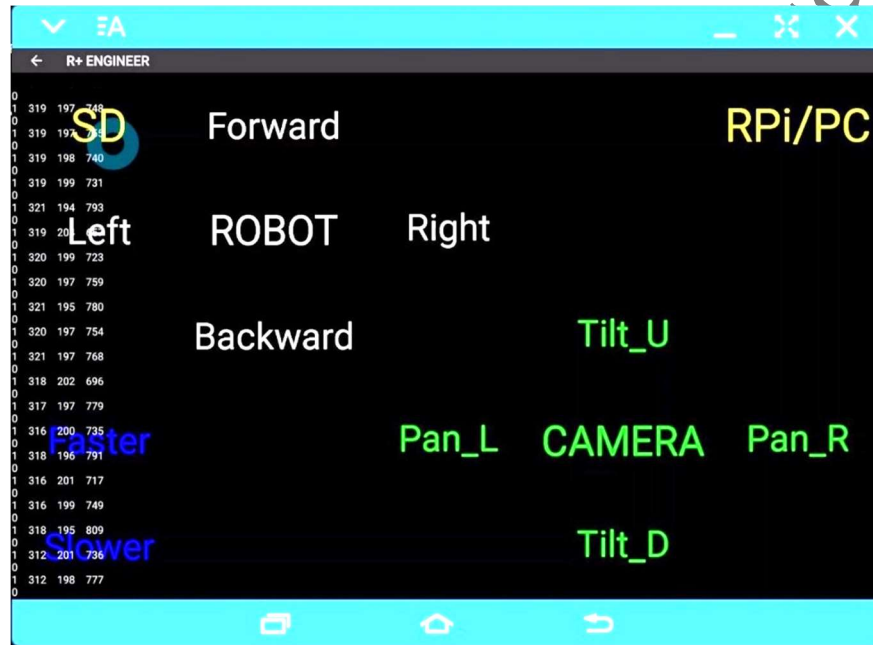


Fig. 4.17 Mobile Display at run time for the Dual Control Project.

4.2.2 Dual Control with Autonomous Obstacle Avoidance

This section describes an “added” feature to the Dual Control capability demonstrated in the previous Section 4.2.1. Fig. 4.1 described how the Left and Right DMS-80s were mounted in front of the E-PTC robot, and in this project, the Pan-Tilt servos would stay at their initial Goal Positions during run time. Therefore, with this configuration, the E-PTC robot could detect a “single” obstacle in 3 ways: right in front of it, or to the left or to the right of it.

The algorithm in this project uses the Dual Control algorithm as previously described in Sub-Section 4.2.1, but only when the DMS-80 sensors report that there is no obstacle “near-by”, that is when both report a sensor value less than a user-set **threshold_1** value (=600, in this project). ...

However, there is one crucial task to execute next before RC commands are allowed back to the operator (i.e., back to the beginning of the Main Endless Loop), and this is illustrated in Fig. 4.29 also:

- Here, the author assumes that the operator still has his/her fingers on the Touch Areas corresponding to the “mistaken” maneuvers, thus the WHILE Loop (Lines 427-428) is used to “wait out” for the operator to release all fingers from the Mobile Touch Screen. The 0 ms delay (Line 428) is used because an “empty-body” loop is not allowed in MicroPython code.
- Lines 431-435 illustrates a more standard coding approach to provide the same run-time result which may be more familiar to some readers (but likely slower).
- As a challenge to the reader, please note that if Parameters **touch1** and **touch2** are used instead in Line 427, this would result in a run-time error (infinite loop): this is left for readers to figure out why.

```

424 # Here, no more obstacle in front - Robot stops and plays music
425 success() ←
426 # Checking if operator has released both finger presses
427 while ((smart.read8(10310) != 0) or (smart.read8(10400) != 0)):
428     delay(0)
429
430 # more explicit procedure, separating read step from check step
431 # touch1 = smart.read8(10310) # Touch Area 1
432 # touch2 = smart.read8(10400) # Touch Area 2
433 # while ((touch1 != 0) or (touch2 != 0)):
434 #     touch1 = smart.read8(10310) # refresh Touch Area 1
435 #     touch2 = smart.read8(10400) # refresh Touch Area 2

```

```

185 def success():
186     global obstacle
187     stop()
188     buzzer.melody(3)
189     buzzer.wait()
190     delay(2000)
191     obstacle = False

```

Fig. 4.29 Part 2 of Obstacle Avoidance Algorithm in “PTC_SA_RCSD_VSR_PC_RPi.py”.

4.3 Using Standard Python on RPi4B and Desktop PC

This Section 4.3 is common to TASK and MicroPython coders and it uses Standard Python. It should be read after either Section 4.1 or 4.2 had been studied. The two projects described in this section implement the overall communications scheme illustrated in Fig. 4.2:

1. The first project is developed for the RPi4B, equipped with the Pi Camera, to work as a Vision Processor with a similar role as the RPi0W provided by ROBOTIS in the ENGINEER Kit 2. The RPi4B can also perform as a Secondary Remote Controller for

- the E-PTC robot using the keyboard from the Desktop PC (with the Mobile Device staying as the Primary Remote Controller).
- The second project is designed to allow the Desktop PC to work as a Central Data Hub receiving various sensors and actuators data from the E-PTC. The full utility of this second project will be realized later in Chapter 5, when it is used to perform Supervisory Control of two “RPi4B+CM-550” based robots: E-PTC and A4WP-H.

4.3.1 RPi4B as Vision Processor/Remote Controller

At the time of writing of this book (Spring 2021), the Pi Camera utilities such as **raspistill**, **raspivid** and the Python **picamera** module were not working with the 64-bit Raspian Beta OS (<https://www.raspberrypi.org/forums/viewtopic.php?f=29&t=213435&p=1839305&hilit=picamera+64+bit#p1839305>). Thus, the author settled on OpenCV, all the while realizing that not all possible hardware performances of the Pi Camera would be achievable with this option.

The Standard Python solution to this project is named “PTC_**RCSD**_Color_Tracker_XY_RPi.py” and it is based on the code named “PTC_**RC**_Color_Tracker_XY.py” which was written for the Windows PC environment and a wired webcam. The program “PTC_**RC**_Color_Tracker_XY.py” was previously described in Sub-Section 3.4.1 in Volume 1 (Thai, 2020), so its details will not be repeated in this Sub-Section 4.3.1 and only new codes required to adapt this code to the RPi4B environment and the Pi Camera will be described further (see Fig. 4.30):

- The PySerial module is used for communications programming between the RPi4B and the CM-550 via their respective USB ports. Line 45 shows that the RPi4B OS assigns the device **ttyACM0** to this USB-to-USB connection (see Appendix B – Section B.1 for more details). The baud rate is set 57.6 Kbps and both read and write time-outs are set to 0 for immediate return upon calling any communication functions such **ser.read()** and **ser.write()** later in the program.

```
(1) 45 ser = serial.Serial("/dev/ttyACM0", 57600, timeout=0, write_timeout=0)

    59 UP_ARROW = 65362
(2) 60 DOWN_ARROW = 65364
    61 LEFT_ARROW = 65361
    62 RIGHT_ARROW = 65363

    125 # setting fps and frame dimensions
(3) 126 vid_cam.set(cv2.CAP_PROP_FPS, 90)
    127 vid_cam.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
    128 vid_cam.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)

(4) 134 vid_frame = cv2.flip(vid_frame, -1) # flip captured frame

    87 # Kernel size used for image processing (21x21 pixels)
    88 # kernel = np.ones((21,21),np.uint8)
(5) 89 element = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(21,21))

    308 #         target_opened = cv2.morphologyEx(target_frame, cv2.MORPH_OPEN, kernel)
    309         target_opened = cv2.morphologyEx(target_frame, cv2.MORPH_OPEN, element)
```

Fig. 4.30 New Codes needed for “PTC_**RCSD**_Color_Tracker_XY_RPi.py”.

Pi Camera FOV @ different WxH & FPS

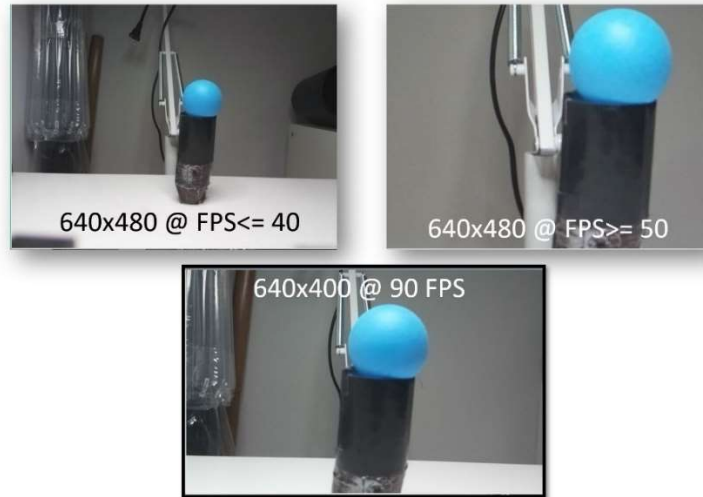


Fig. 4.31 Pi Camera's Field of View (FOV) changes depending on chosen combination of frame rate and frame size.

4.3.2 Desktop PC as Central Data Hub

In Sections 4.1 and 4.2 (see Fig. 4.4), the TASK tool called “Program Monitor Output” was used to display PRINTLN outputs of the IMU's Yaw data from the CM-550 directly to the Desktop PC, via a BT-210 connected to the CM-550's UART Port. In this Sub-Section 4.3.2, a self-standing Python program is developed for this task, its name is “PTC_Data_Central_PC.py” and it is derived from the program named “PTC_MB_DBT_Central.py” described in Sub-Section 3.4.4 of Volume 1 (Thai, 2020). Later in Chapter 5, “PTC_Data_Central_PC.py” will be enhanced to allow the PC to become a Supervisory Controller of two robots: E-PTC and A4WP-H.

```

100 if ((key_1 == p) or (key_2 == p)): # Toggling on Key p (i.e. PTC)
101     if (robot_1 == False):
102         img_1 = cv2.imread(im_PTC_ON)
103         cv2.imshow(window_1, img_1)
104         robot_1 = True
105         btser1.reset_input_buffer()
106         time.sleep(0.1)
107     else:
108         img_1 = cv2.imread(im_PTC_OFF)
109         cv2.imshow(window_1, img_1)
110         robot_1 = False

```



Fig. 4.34 Key “p” flip-flops **robot_1** to TRUE/FALSE and **img_1** to **im_PTC_ON/OFF** in “PTC_Data_Central_PC.py”.

4.4 Using C/C++ on RPi4B and Desktop PC

This Section 4.4 is also common to TASK and MicroPython coders and it uses Standard C/C++. The projects described in this section implement the overall communications scheme illustrated in Fig. 4.2:

1. The first C++ project was developed for the RPi4B, equipped with the Pi Camera, to work as a Vision Processor with a similar role as the RPi0W provided by ROBOTIS in the ENGINEER Kit 2. The RPi4B would also be able to perform as the Secondary Remote Controller for the E-PTC robot using the Desktop PC via VNC Viewer, of course with the Mobile Device staying as the Primary Remote Controller.
2. The second C++ project is designed to allow the Desktop PC to work as a Central Data Hub receiving various sensors and actuators data from the E-PTC. The full utility of this second project will be realized later in Chapter 5, when it is used to perform Supervisory Control of two “RPi4B+CM-550” based robots: E-PTC and A4WP-H.

4.4.1 RPi4B as Vision Processor/Remote Controller

In the author’s earlier experiments with the RPi4B, the C/C++ route always gave the best performances over the Python route, so the author was surprised and disappointed at the actual poor performances obtained when using C/C++ via Code::Blocks V.17.2, and having **the Mobile Device as Main Remote Controller** (see video at https://www.youtube.com/watch?v=dt29_zM-eWA). As the Raspian 64-bit OS is in its Beta phase currently (September 2021), the author does not know if the following issues reported here will be resolved in the future or not.

On the RPi4B and for C/C++ programming, the user has a choice between using the ROBOTIS ZigBee SDK or the BOOST ASIO Library, when only 1 communication port is needed between the RPi4B and the CM-550 (which is the case for this first project):

1. Programs using the **ttyACM0** Port, i.e., via the USB cable, had the following problems:
 - The ZigBee SDK can create codes that would compile with no errors, but at run time these codes would report that they could not connect through **ttyACM0**.
 - The Boost ASIO Library can create codes that would compile and execute with no errors, but their run-time performances were very poor through **ttyACM0**.
2. Programs using **ttyS0** (i.e., plain 3-wire cable) or **ttyUSB0** (actual LN-101 on RPi4B USB Port), had the following issues:
 - The ZigBee SDK can create codes that would compile and run with no errors, but their run-time performances were very poor.
 - The Boost ASIO Library can create codes that would compile and execute with no errors, but their run-time performances were also very poor. ...

The author inferred from all the above results that the ENGINEER App must have been requiring much computing and communications resources from the CM-550’s BLE Port such that it neglected the UART message traffic between the RPi4B and the CM-550 (see related issue in Sub-Section 6.2.2). But when using the BLE Port for simpler PRINTLN commands requiring less resources, then the CM-550 can reroute more resources to the UART message flow and therefore improve object tracking performances. **Therefore, if the ENGINEER App needs to be used in the reader’s project, the best development environment is Python.**

4.4.2 Desktop PC as Central Data Hub

For the Central Data Hub project in C++, the author chose the **tttACM0** option for the RPi4B/CM-550 connection via USB ports, and the **BT-210 via UART** to connect between the CM-550 and the PC, i.e., the same hardware set up used in the Python solution described in Sub-Section 4.3.2. Thus, on the RPi4B the author used the previous program “PTC_RC_Color_Tracker_XY_RPi_BST.cpp”, and on the CM-550, he also used the previous program “PTC_RC_VSR_PCDH_RPi.tsk3/py”. A reminder that the program “PTC_RC_VSR_PCDH_RPi.tsk3/py” has an IF-ELSE structure designed to trigger the “Yaw-Angle” event only after a given **Count** (=100) of repetitions performed by the Main Endless Loop. ...

The run-time performance of this C++ program was much better than its Python equivalent from the previous Sub-Section 4.3.2. But Fig. 4.37 showed that the (X, Y, Area) Remocon packets were sent from the RPi4B to the CM-550 at a mere 57.6 Kbps baud rate and consequently the Pan-Tilt platform was not quite responsive to the author’s liking, yet! Fortunately, in late Spring 2021, ROBOTIS released a Dynamixel HAT (Hardware Attached on Top) for the RPi4B’s 40-pin GPIO bus which would allow the RPi4B to directly control the XL430 actuators at the much faster Dynamixel baud rate of 1 Mbps. The next section 4.5 details the integration issues to be resolved for this DXL-HAT to work proficiently between the RPi4B and the CM-550.

4.5 Using DXL-HAT and Dynamixel SDK

First, there was the issue of how to mount the DXL-HAT onto the E-PTC robot as configured so far (see Fig. 4.18). Section B.9 has more details on the pros and cons of different configurations tested out by the author which are not repeated here (including the one using the U2D2 module). The “final” configuration used for the current project is shown in Fig. 4.39. It uses a passive-cooling case for the RPi4B as the most compact (but not the lightest) configuration available at the time of writing of this book.

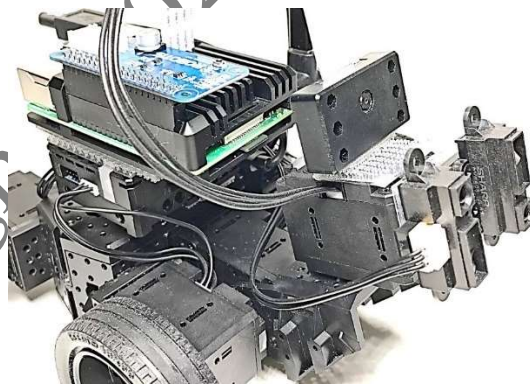


Fig. 4.39 Passive-Cooling case for the RPi4B and DXL-HAT.

The second issue was how to “appropriately” control the XL430s, IMU, DMS-80s and Pi Camera used by the E-PTC robot. This issue turned out to be more multi-faceted than the author expected when he first worked on this issue.

Section 1.4 in Vol. 1 described the main features of the ROBOTIS Dynamixel Network used in their robotics kits. This network is hierarchical, meaning that there can be only ONE Top Controller in this DXL Network. However, the author wanted/needed a “split” system whereas the Pan-Tilt platform and the Pi Camera are to be controlled by the RPi4B, while the Wheeled Platform and IMU+DMS-80 sensors remained under the control of the CM-550.

This “split” Dynamixel Control scheme is conceptually described in Fig. 4.40:

- A single Li-Po battery (and a 12V-5V converter) provides power to both “CM-550” and “RPi4B” Dynamixel Networks to provide a common electrical ground level for all devices which is **mandatory** for all communications packets to propagate “reliably” within this split network.
- The CM-550 controls DXLs (1, 2, 3, 4) and the IMU+DMS-80 sensors as previously done, via TASK and MicroPython programming. The CM-550 also maintains BT contact with a Desktop PC via its UART Port and a BT-210. The CM-550 additionally can receive Remocon packets from the RPi4B via a USB cable (Micro-USB Port on the CM-550 and **ttyACM0** Port on the RPi4B).

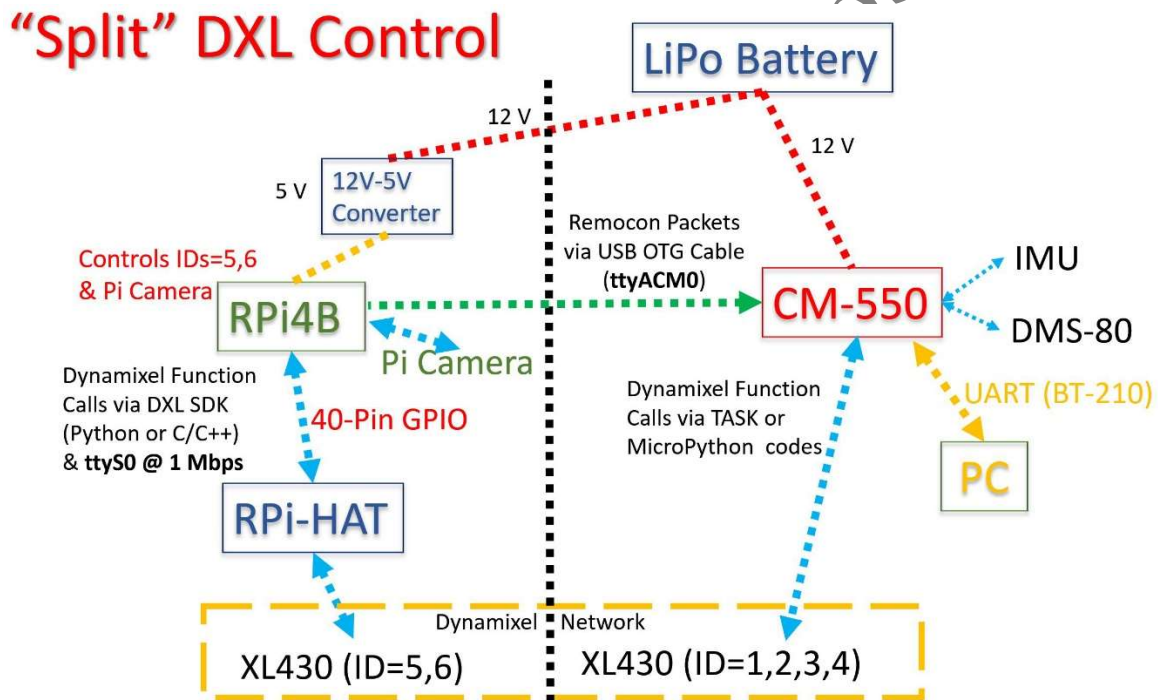


Fig. 4.40 “Split” Dynamixel Control Scheme used for the E-PTC Robot.

Once this Split Control scheme was implemented, the author started to check the performances of the DXL-HAT with the Dynamixel SDK, using Python and C++ examples provided by ROBOTIS for the RPi series. Unfortunately for Python fans, “sync_read_write” procedures worked reliably **only** at 115.2 Kbps or lower in Python codes (at least on the author’s RPi4Bs). This issue could be resolved in future releases of the 64-bit Raspian OS which is still in the Beta Phase at the time of writing for this book (September 2021).

However, using C++ codes, the author could get the DXL-HAT to perform reliably at 1 Mbps (i.e., at Dynamixel Bus rate). **Thus, for the remainder of this Section 4.5 only a C++ project is showcased.**

4.5.1 RPi4B as Vision Processor/Remote Controller

This C++ project aimed at integrating previous features such as Manual Remote Control (RC), Smart Avoider (SA) and Visual Servo Ranging (VSR), but they will be organized differently. The E-PTC robot now has two operating modes:

1. In its Manual RC mode, the operator can use the PC keyboard (via VNCViewer's control of the RPi4B) to perform the standard UDLR maneuvers for the robot using "commType 0" Remocon packets. However, the CM-550's Smart Avoider feature will take over if the operator drives the robot into an obstacle on purpose. The Pi Camera is not used in this mode.
2. In its Autonomous Color Object Tracking mode, the RPi4B directly controls the Pan-Tilt platform (servos 5 and 6) to track the object using its location data provided by the Pi Camera, while sending "directional" Remocon packets to the CM-550 which controls servos 1 through 4 to perform the required wheel maneuvers for additional tracking actions. In this mode, the CM-550 should not use its Smart Avoider feature (i.e., Object Avoiding) as it would contradict with its current Object Tracking mission.

On the RPi4B side, the code solution is called "IDN_PTC_SA_RC_Color_Tracker_RPi.cpp" and its CM-550's co-code solution is called "IDN_PTC_SA_RC_VSR_PCDH_RPi.tsk3".

Let us first look at the main features of "IDN_PTC_SA_RC_Color_Tracker_RPi.cpp" in Figs. 4.42 through Figs. 4.46, illustrating the setup and usage of the Dynamixel SDK in a C/C++ environment.

Fig. 4.42 describes the initialization steps needed to use the Dynamixel SDK:

- Lines 32-34 define the XL430's Control Table addresses that are used in this project: TORQUE_ENABLE, GOAL_POSITION and PRESENT_POSITION.
- Lines 36-37 define the Data Byte Lengths (4 bytes) used for GOAL_POSITION and PRESENT_POSITION, which are needed when setting up for a "sync_write" or "sync_read" procedure.
- Line 43 shows that the DXL-HAT is set to 1 Mbps baud rate and Line 44 shows that the Raspian OS attached it to Device "ttyS0" via GPIO connection. The reader/user may have to change the ownership rights on **ttys0** once (before running the program "IDN_PTC_SA_RC_Color_Tracker_RPi.cpp" for the first time), using a **bash** Terminal and issuing the following command:
 - **sudo chmod a+rw /dev/ttyS0**
- Lines 48-56 list other variables defined by the author for this project.

```

31 // Control table address
32 #define ADDR_X_TORQUE_ENABLE 64
33 #define ADDR_X_GOAL_POSITION 116
34 #define ADDR_X_PRESENT_POSITION 132
35 // Data Byte Length
36 #define LEN_X_GOAL_POSITION 4
37 #define LEN_X_PRESENT_POSITION 4
38 // Protocol version
39 #define PROTOCOL_VERSION 2.0
40 // Default settings
41 #define DXL1_ID 5
42 #define DXL2_ID 6
43 #define BAUDRATE 1000000
44 #define DEVICENAME "/dev/ttyS0"
45 #define TORQUE_ENABLE 1
46 #define TORQUE_DISABLE 0
47 #define DXL_MOVING_STATUS_THRESHOLD 20

48 int dxl_comm_result = COMM_TX_FAIL;
49 bool dxl_addparam_result = false;
50 bool dxl_getdata_result = false;
51 int IDs[2] = { 5, 6 };
52 int dxl_goal_position[2] = { 2048, 2048 };
53 uint8_t dxl_error = 0;
54 uint8_t param_goal_position[4];
55 int32_t dxl_present_position[2] = { 0, 0 };
56 bool read_present_positions_OK = false;

57 // Initialize PortHandler & PacketHandler instances
58 dynamixel::PortHandler* portHandler = dynamixel::PortHandler::getPortHandler(DEVICENAME); // ttyS0
59 dynamixel::PacketHandler* packetHandler = dynamixel::PacketHandler::getPacketHandler(PROTOCOL_VERSION);
60 // Initialize GroupSyncWrite & GroupSyncRead instances
61 dynamixel::GroupSyncWrite groupSyncWrite(portHandler, packetHandler, ADDR_X_GOAL_POSITION, LEN_X_GOAL_POSITION);
62 dynamixel::GroupSyncRead groupSyncRead(portHandler, packetHandler, ADDR_X_PRESENT_POSITION, LEN_X_PRESENT_POSITION);

```

Fig. 4.42 Initialization Steps for DXL-SDK in “IDN_PTC_SA_RC_Color_Tracker_RPi.cpp”.

```

709 // Preparing TxData (comm_Type=1) for possible UDLR Remocon commands to adjust wheeled platform
710 TxData = (1 << 14); // prepare comm Type=1 packet
711 if ((target_x > 0) && (target_x <= X1)) // turn left
712 {
713     m_LEFT = 1;
714     TxData |= (m_LEFT << 2); // VRC Button LEFT added with bit-wise OR
715     data_2_packet(TxData);
716     write(ser1, buffer(TxD_packet, 6)); // send LEFT packet
717     m_LEFT = 0;
718 }
719 else if ((target_x >= X3) && (target_x <= Res_Width)) // turn right
720 {
721     m_RIGHT = 1;
722     TxData |= (m_RIGHT << 3); // VRC Button RIGHT added with bit-wise OR
723     data_2_packet(TxData);
724     write(ser1, buffer(TxD_packet, 6)); // send RIGHT packet
725     m_RIGHT = 0;
726 }

```

Fig. 4.51 Programming steps used when sending UDLR Remocon packets when (track_mode == 1) in “IDN_PTC_SA_RC_Color_Tracker_RPi.cpp”.

4.5.2 CM-550 as Smart Avoider

Let us next look at the companion TASK code “IDN_PTC_SA_RC_VSR_PCDH_RPi.tsk3”, but only at the main features/changes implemented to deal with the use of “comm_Type”.

The overall Obstacle Avoidance algorithm previously described in Section 4.1.2 is still followed in this project, except that the execution code for the RC commands and the Obstacle Avoidance procedure have been moved into their own function to achieve a “cleaner” look for the Main End-less Loop.

4.6 Desktop PC as Supervisory Controller

In this last project for Chapter 4, the IDN/DXL-HAT version of the E-PTC is used (see Fig. 4.39) and the goal is to set the Desktop PC as a Supervisory Controller, meaning that the PC can override the RPi4B’s current commands to the CM-550.

4.6.1 Revising Robot Communications Scheme

The original overall communications scheme (as shown in Fig. 4.2) had to be revised into Fig. 4.55 as several communication issues had to be solved differently than previously planned:

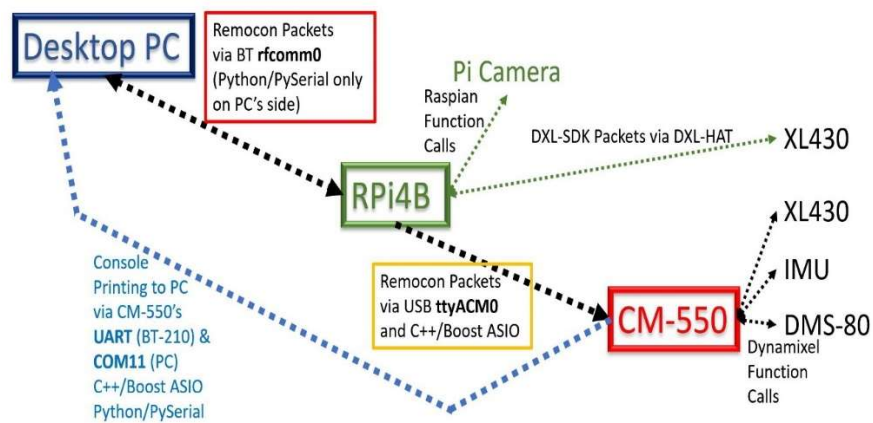


Fig. 4.55 Revised Communications Scheme used for “PC as Supervisory Controller” Project.

4.6.2 Supervisory Control Demonstration for E-PTC

Fig. 4.59 shows the overall mechanical and communications configurations for the E-PTC robot in its IDN version. Please view/review Section B.3 (Appendix B) for setup information for the BT connection between the PC and the RPi4B via the **rfcmm0** device on the RPi4B.

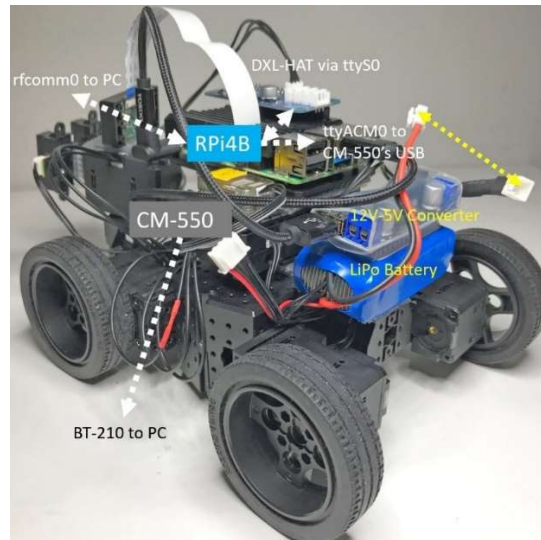


Fig. 4.59 Mechanical and Communications Configurations of E-PTC (IDN version).

In summary for this Supervisory Control project, there are 4 programs that need to be executed in a coordinated fashion:

- On the PC, “SC_PTC_A4WP_2BT_Central_BST.py” and “PTC_Data_Central_PC.cpp”.
- On the RPi4B, “IDN_PTC_SA_SC_Color_Tracker_RPi.cpp”.
- On the CM-550, “IDN_PTC_SA_RC_VSR_PCDH_RPi.ts3/py”.

There is a 5-step procedure that must be followed when demonstrating this Supervisory Control project (Fig. 4.60):

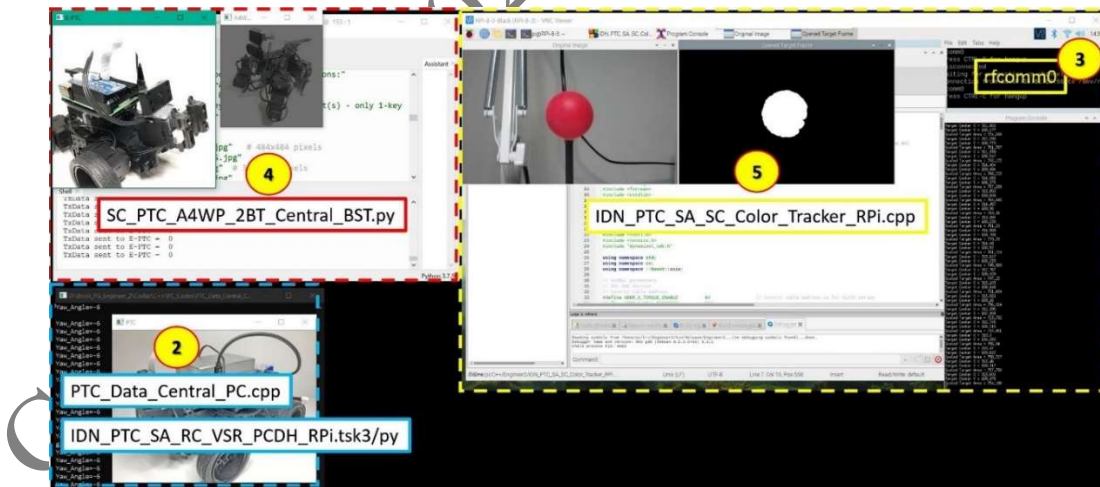


Fig. 4.60 Runtime Screen Capture for project “PC as Supervisory Controller” for the E-PTC robot.

Chapter 5: Quadruped and its Variants with RPi4B

This Chapter 5 builds on concepts explained and solutions described in Chapter 4, thus the reader is encouraged to read Chapter 4 if not done so.

Fig. 5.1 shows the “Enhanced” Quadruped (E-QUAD) used for projects described in this Chapter 5:

- An RPi4B, with 8 GB RAM and Pi Camera running on 64-bit Raspian, is used as a “Co-Controller” to the CM-550, as per ROBOTIS Dynamixel Network Paradigm (see Section 1.4 of Volume 1 (Thai, 2020-a), in which the CM-550 is the “Top-Controller”. This RPi4B shares into the CM-550’s LiPo battery using a 12V-to-5V converter. The RPi4B communicates with the CM-550 via an OTG (or plain) USB cable (Port `tttyACM0`). The RPi4B also communicates wirelessly with a Desktop PC via its Bluetooth Controller as Port `rfcomm0` (see Appendix B for more details about setting up the RPi4B and CM-550, hardware-wise and software-wise).

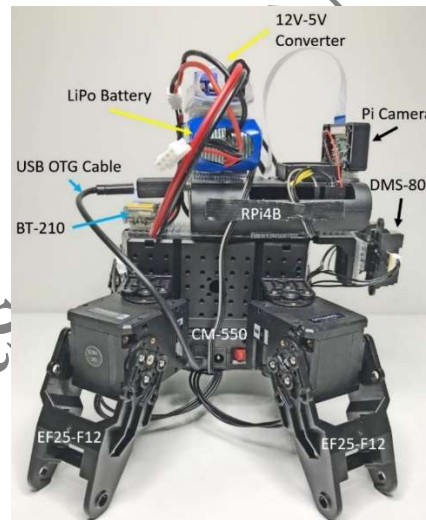


Fig. 5.1 “Enhanced” Quadruped with RPi4B and Pi Camera.

- The CM-550 is further equipped with two DMS-80 sensors via its GPIO Ports 3 and 4 and can communicate with the Desktop PC via a BT-210 connected to its UART Port.
- The reader also needs to make sure that the TASK App is **V. 3.1.5** or above for the example codes in this Chapter to work properly.
- The MOTION component of TASK 3 will be used more extensively in Chapter 5, thus the reader is encouraged to review the “TASK 3 Programming Curriculum” manual

from ROBOTIS. For a more in-depth description and practicum of the MOTION tool, please read Chapter 4 of the author's MINI book (Thai, 2020-b).

Mechanically, this E-QUAD robot is basically the same as the Standard Quadruped robot described in the ROBOTIS "TASK 3 Programming Curriculum" manual available at this web link http://en.robotis.com/model/login.php?url=/pdf_project/register.php# (the interested user needs to register his/her CM-550 Serial Number with ROBOTIS at this web link to be able to download this "free" document). But Fig. 5.1 shows that the robot's top area is now quite "crowded" with the RPi4B, the power coupling hardware and the Pi Camera.

Thus, in this Chapter 5, the E-QUAD robot also has its own version of the E-PTC's "Dual-Control/VSR" and "Smart Avoider" projects but as a WALKING robot (in both TASK and MicroPython versions of course). But a more interesting development is to add 4 XL-430s set in wheel mode to the legs of E-QUAD which then becomes more of an "Articulated 4-Wheel Platform" (A4WP). This A4WP robot can then be configured into several wheel-based travel modes (see Figs. 5.3 and 5.4).

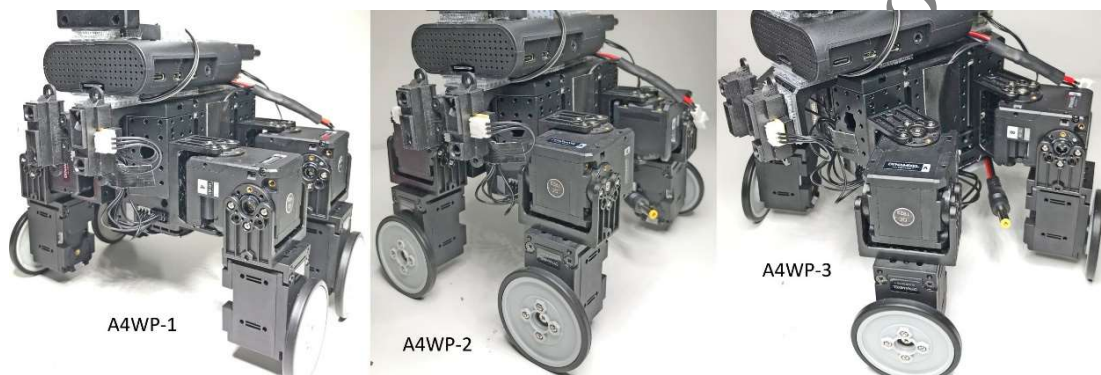


Fig. 5.3 Three Possible Modes for Articulated 4-Wheel Platform (A4WP) Robot.



Fig. 5.4 Hybrid Mode A4WP-H, with "walking" front legs and "rolling" rear legs.

5.1 Using TASK

The first two TASK projects are described in Sub-Section 5.1.1 and 5.1.2, and both used a USB cable to link the RPi4B to the CM-550's micro-USB Port, and a BT-210 to link the Desktop PC to the CM-550's UART Port:

1. The first one (Sub-Section 5.1.1) establishes a Dual Control framework so that the CM-550 can be controlled by either the RPi4B and a Desktop PC, or both at the same time. This TASK solution is named "QUAD_DRC_VSR_PC_RPi.tsk3", to be used with the Motion file named "QUAD_DRC_VSR_PC_RPi.mtn3".
2. The second one (Sub-Section 5.1.2) adds an Autonomous Obstacle Avoidance feature to the previous Dual Control framework, and this TASK solution is named "QUAD_SA_DRC_VSR_PC_RPi.tsk3", and still using "QUAD_DRC_VSR_PC_RPi.mtn3".

In these example TASK codes, if the reader does not have access to an RPi SBC, he/she can use a USB cable to connect a Desktop PC to the CM-550's micro-USB port, then the operator can at least check out the Manual RC features implemented in these programs.

5.1.1 Dual Control from RPi4B/Desktop PC

The author at first used the Motion Units that ROBOTIS created for its demonstration codes for the Standard Quadruped but he found they moved the robot "too much" so that the color target could shift out of the Pi Camera's viewport with just one single Motion Unit being executed (especially the "turn" moves). To reduce the U-D-L-R "Motion Amplitudes", a simple modification procedure is shown in Fig. 5.5 for the Motion Unit labeled "09-up" (i.e., robot walking forward) and only for its first Motion Frame (defined at time=117 ms on the Timeline). The reader can readily see that the Goal Position values in degrees had been reduced in half for the Servo IDs = 1, 3, 5, and 7, while the other servos' settings remained the same. These types of modifications "roughly" reduce the U-D-L-R "Motion Amplitudes" by half.

1. "To Wait or Not To Wait for Motion Status Changes?":

Fig. 5.6 shows the standard "Motion Play" procedure that ROBOTIS uses in its provided demonstration codes written for the Standard Quadruped (and other ENGINEER 1 and 2 robots also). This procedure would start with taking the user's input (such as from a Touch Area – Line 174) which is then matched with a given Motion Unit (such as Motion 15 – Line 177). The next step would "play" this given Motion Unit (Line 183) which would **take the robot some time to perform**. This Play step would also be **immediately followed** by a Waiting Function (Line 184).

2. "Motion Speed Changes"

This issue is easily solved as the approach used in the E-PTC's projects still applies (see Fig. 4.9) with one additional step (see Lines 353 and 366 of Fig. 5.8).


```

344 FUNCTION Increase_Speed
345 {
346   IF ( Speed < 150 )
347   {
348     Speed = Speed + 10
349     IF ( Speed >= 150 )
350     {
351       Buzzer Timer = 0.2sec
352       Buzzer Index = Sol (10)
353       Motion Speed = Speed
354     }
355   }
356 }
357 FUNCTION Decrease_Speed
358 {
359   IF ( Speed > 20 )
360   {
361     Speed = Speed - 10
362     IF ( Speed <= 20 )
363     {
364       Buzzer Timer = 0.2sec
365       Buzzer Index = Re (17)
366       Motion Speed = Speed
367     }
368   }
369 }

```

Fig. 5.8 “Motion Speed” change procedure used in this book’s “walking” projects.

3. “Robot Body Tilting to aim Pi Camera”

Figure 5.2 illustrates the issue encountered by the E-QUAD robot when it wants to aim the Pi Camera down, whereas it has to spread its front legs outward so that its front-end is lower relative to its rear-end. Of course, the reverse procedure is performed when it needs to aim the Pi Camera up. This body-tilting feature needs to be effective for all Motion Units of course and it can be achieved by setting the ADJUSTED OFFSETs of the appropriate servos (ID = 2, 4, 6 and 8 in this case).

5.1.2 Dual Control with Autonomous Obstacle Avoidance

Adding the Autonomous Obstacle Avoidance capability to the E-QUAD required only minimal efforts when starting from the previous code developed for the E-PTC, i.e., “PTC_SA_RCSD_VSR_PC_RPi.tsk3” and combining it with the Walking procedures implemented in “QUAD_DRC_VSR_PC_RPi.tsk3”, to obtain the solution “QUAD_SA_DRC_VSR_PC_RPi.tsk3”.

5.1.3 Articulated Four-Wheeled Platform (A4WP-123)

To convert the E-QUAD as shown in Fig. 5.1 into an “Articulated 4-Wheel Platform” (A4WP-123), the author removed the Frame Part EF25-F12 from all four legs and replaced them with the “Claw-Wheel” module (see Fig. 5.13), one for each leg, which could be constructed from:

1. One XL430-W250-T actuator.
2. One EF25-F14 frame part and one EF25-F24 frame part assembled perpendicularly to each other.
3. One Thin Wheel and Rubber Tire from the BIOLOID FP04-F13/F14 Set, available at this web link (<https://www.robotis.us/fp04-f13-f14-4set/>). The author preferred this thinner wheel over the thicker wheel that comes with the ENGINEER Kit 2, because in certain configurations the thinner wheel is more suitable as a “claw” to enhance “walking” maneuvers (see Fig. 5.4).

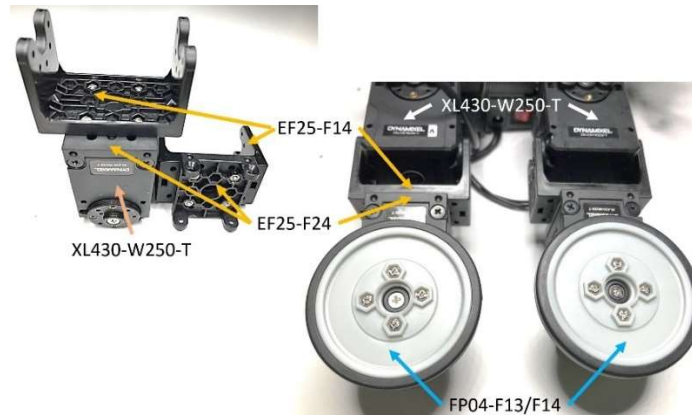


Fig. 5.13 “Claw-Wheel” Module used in A4WP based robots.

5.1.4 Hybrid A4WP (A4WP-H)

Although the Hybrid A4WP (A4WP-H) uses the same parts as the original A4WP-123 robot, Fig. 5.14 shows that their assembly configurations are quite different from each other. The A4WP-123 robot uses a central symmetry theme for its 4 legs, while the A4WP-H uses an anti-symmetry theme for its 4 legs: for example, frame parts EF25-F14 and EF25-F24 can be in-line or perpendicular to each other, and the wheels can be mounted on the outside or inside of the actuators.

Furthermore, the A4WP-H robot has “walking” frontal legs and “rolling” rear legs. The interesting feature is that the “Walking” Motion Units created for the Original Quadruped’s 4 legs can still be used with the A4WP-H robot via an **undocumented Parameter** located at Address **1016 + ID** that can “decouple” the rear legs from the effects of the Motion Units that were “originally” moving them. The solution to this project is named “A4WP-H_DRC_VSR_PC_RPi.tsk3/mtn3”. In this example TASK code, if the reader does not have access to an RPi SBC, he/she can use a USB cable to connect a Desktop PC to the CM-550’s micro-USB port, then the operator can at least check out the Manual RC features implemented in these programs.

5.2 Using MicroPython

This section is written for MicroPython coders who did not read Section 5.1 (but if the reader did study Section 5.1 before getting here, please bear with some “unavoidable” information repetitions).

Section 5.2 describes MicroPython projects involving the E-QUAD robot (Fig. 5.1) and its variants: A4WP-123 (Fig. 5.3) and A4WP-H (Fig. 5.4). Two Motion Control techniques will be showcased:

1. **Using MTN3 Motion files** – In Sub-Section 5.2.1, the E-QUAD robot is used to demonstrate this technique which has its origin from its TASK equivalent. Two MicroPython projects are described: the first project establishes a Dual Control Framework (Manual RC from RPi4B or CM-550 Autonomous Color Tracking using Pi Camera attached to RPi4B), and the second one added an Obstacle Avoidance feature to the first. Their respective solutions are named “QUAD_DRC_VSR_PC_RPi.py” and “QUAD_SA_DRC_VSR_PC_RPi.py”.

2. **Using Time Control and Motion Arrays** – In Sub-Section 5.2.2, the second E-QUAD project “QUAD_SA_DRC_VSR_PC_RPi.py” is re-written to use Time-Control, as an example for how to convert between the two control techniques (“TC_QUAD_SA_DRC_VSR_PC_RPi.py”). Next the Time-Control technique is further applied to the A4WP-123 and A4WP-H robots, resulting in the solutions “A4WP_DRC_VSR_PC_RPi.py” and “A4WP-H_DRC_VSR_PC_RPi.py”.

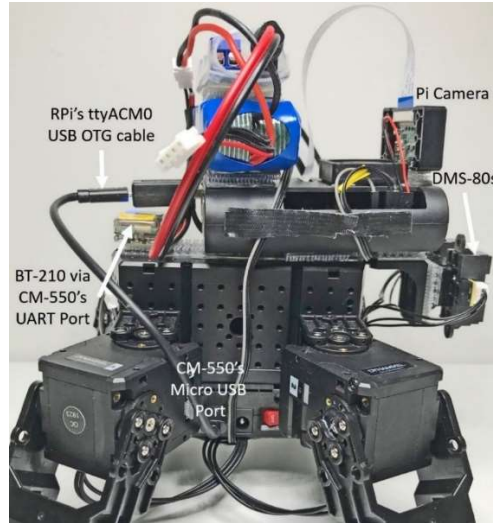


Fig. 5.27 Communications setup between RPi4B and CM-550 in Section 5.2.

5.2.1 Using MTN3 Motion Files

The Motion file used in Sub-Section 5.2.1 is “QUAD_DRC_VSR_PC_RPi.mtn3”, which is the same one described at the beginning of Sub-Section 5.1.1 (please review if needed).

The solution for the Dual Control Framework project is named “QUAD_DRC_VSR_PC_RPi.py” and only the main programming features will be further elaborated on in this Sub-Section.

Fig. 5.29 describes the procedures used in Part 1 of the Main Endless Loop (Main Algorithm):

- Line 177 checks if the CM-550 has received a new Remocon packet from the PC. If this event happens, the Function **rc.received()** would return **TRUE**, then this big IF structure is entered at Line 178, where Function **rc.read()** extracts from the Remocon packet the actual data component and saves it in Parameter **data_in**. Parameter **data_in** has multiple components embedded in it and this structure was previously described in Sub-Section 3.4.1 of Thai (2020-a) (please review if needed).
- Line 179 extracts Parameter **comm_Type** from **data_in** and shifted it right 14 bits (to its proper bit position).
- Next, starting at Line 181, the steps are to extract out current information on various **RC BUTTONs** “U-D-L-R-1-2-3-4-5-6” (Line 182 to Line 191).
- If **data_in** turns out to be zero, this means that the operator just releases all keys on the keyboard attached to the RPi4B (Line 192), then the robot needs to stop (Line 193).

- Function **stop()** is displayed at the bottom right of Fig. 5.29 (Lines 41-43). At the time of writing of this book, the Function **motion.status()** was not working properly for the author, but a direct call to its address was working, thus the use of **etc.read8(68)** in Line 42. This IF conditional statement checks to see if the robot MOTION STATUS is FALSE (meaning that it is not performing any motion). So, if the robot happens to be still, then it is commanded to play MOTION 1 (Line 43) which will put the robot into its Ready pose.

```

174 while True:
175     data_in = 0
176     comm_Type = -1
177     → if (rc.received() == True):
178         data_in = rc.read()
179         comm_Type = (data_in & 0xc000) >> 14
180
181     → if (comm_Type == 0): # Processing UDLR123456 packets
182         forward = data_in & rc.BTN_U
183         backward = data_in & rc.BTN_D
184         left = data_in & rc.BTN_L
185         right = data_in & rc.BTN_R
186         tilt_up = data_in & rc.BTN_1
187         tilt_down = data_in & rc.BTN_3
188         slide_left = data_in & rc.BTN_2
189         slide_right = data_in & rc.BTN_4
190         faster = data_in & rc.BTN_5
191         slower = data_in & rc.BTN_6
192         if (data_in == 0):
193             stop()

```

```

41 def stop():
42     if (etc.read8(68) == 0): # motion.status()
43         motion.play(1) # Ready pose

```

Fig. 5.29 Part 1 of Main Endless Loop in “QUAD_DRC_VSR_PC_RPi.py”.

Fig. 5.30 describes the procedures used in Part 2 of the Main Endless Loop:

- If **data_in != 0**, the ELSE branch on Line 194 is taken, and a big IF-ELSE-IF structure is processed next. Thus, depending on which RC Button among UDLR123456 is pushed at runtime (see Fig. 5.29), a specific action will be executed next – for example **go_forward()** or **right_slide()**.
- For example, if **Up Button** is pushed, Parameter **forward** would be positive, and Function **go_forward** will be called and thus MOTION 2 will be played (Line 47).

```

194     else:
195         # Standard RC control for UDLR
196         if (forward > 0):
197             go_forward()
198         elif (backward > 0):
199             go_backward()
200         elif (left > 0):
201             turn_left()
202         elif (right > 0):
203             turn_right()
204         elif (slide_left > 0): #
205             left_slide()
206         elif (slide_right > 0):
207             right_slide()
208         elif (tilt_up > 0): # RC
209             → up_tilt()
210         elif (tilt_down > 0):
211             → down_tilt()
212         elif (faster > 0): # Char
213             → increase_speed()
214         elif (slower > 0):
215             → decrease_speed()

```

```

45 def go_forward():
46     if (etc.read8(68) == 0):
47         motion.play(2) # Crawl Forward
48
49 def go_backward():
50     if (etc.read8(68) == 0):
51         motion.play(3) # Crawl Backward
52
53 def left_slide():
54     if (etc.read8(68) == 0):
55         motion.play(4) # Slide Left
56
57 def right_slide():
58     if (etc.read8(68) == 0):
59         motion.play(5) # Slide right

```

Fig. 5.30 Part 2 of Main Endless Loop in “QUAD_DRC_VSR_PC_RPi.py”.

Fig 5.31 provides a closer look at what happens when Function **up_tilt()** is called:

- Line 81 shows that an “up_tilt” operation essentially means to increase Parameter **tilt_offset** by 10, each time that Function **up_tilt()** is called.
- Lines 82-84 make sure that **tilt_offset** will not exceed **hi_tilt** (= 450) (Lines 82-83). When this upper limit is reached, an audible alarm will be heard (Line 84).
- Lines 85-86 effectively “do nothing” (Line 86) as long as **Address 68** (i.e., **Motion.Status()**) reads 1 (= TRUE) – in other words when the robot was moving. This step is to prevent the code from setting servo offsets if the servos are still in motion.
- When it is assured that the robot is no longer moving, this WHILE Loop will exit to Line 87 and Function **set_tilt_offset()** is called.

Fig 5.31 also shows the steps implemented when Function **set_tilt_offset()** is called:

- As an example, let us say that **up_tilt()** was called first, then it called **set_tilt_offset()**.
- In this case, Global Parameter **tilt_offset** most likely is positive so the ELSE block (Lines 97-101) would apply. Thus, a **0 Offset** is set to Servos 2 and 4 (corresponding to the robot’s front legs) and a **negative Offset** is set to Servos 6 and 8 (corresponding to the robot’s rear legs). Physically, the front legs would then keep their “normal gait” when performing prescribed motions, while the rear legs would “spread out” more when performing their motions. Therefore, the net result is the lowering of the rear body with respect to the front body, so the robot would be “tilting up”!

- Lastly, Address 199 is checked to make sure that these Offset values are properly written into their memory locations on the CM-550, before exiting Function **set_tilt_offset()**. Regarding Line 103, the author did not see any outwardly difference in the robot behavior whether Line 103 is enabled or not.

```

79 def up_tilt():
80     global tilt_offset, hi_tilt(= 450)
81     tilt_offset += 10
82     if (tilt_offset > hi_tilt):
83         tilt_offset = hi_tilt
84         alarm()
85     → while(etc.read8(68) == 1):
86         delay(0)
87     → set_tilt_offset()
88
89 def set_tilt_offset():
90     global tilt_offset
91     print('Tilt_Offset = ', tilt_offset)
92     if (tilt_offset <= 0):
93         etc.write16(268, tilt_offset) # (Servo 2)
94         etc.write16(272, tilt_offset) # (Servo 4)
95         etc.write16(276, 0) # set tilt_o (Servo 6)
96         etc.write16(280, 0) # set tilt_o (Servo 8)
97     else:
98         etc.write16(268, 0) # set tilt_o (Servo 2)
99         etc.write16(272, 0) # set tilt_o (Servo 4)
100        etc.write16(276, -tilt_offset) # (Servo 6)
101        etc.write16(280, -tilt_offset) # (Servo 8)
102
103 # etc.write8(199, 2)
104     → while (etc.read8(199) != False):
105         delay(0)

```

Fig. 5.31 Functions **up_tilt()** and **set_tilt_offset()** in “QUAD_DRC_VSR_PC_RPi.py”.

Let us look next at how the robot handles Object Tracking data coming over from the RPi4B (Figs. 5.32 and 5.33). Each time that the RPi4B is successful in tracking down a Color Object via its Pi Camera, it sends 3 consecutive Remocon packets, each for the following types of data (review Sub-Section 3.4.1 of Thai (2020-a) if needed):

- object_X: pixel column location of Barycenter of Color Object.
- object_Y: pixel row location of Barycenter of Color Object.
- Object_Area: pixel area of Color Object.

At the bottom right of Figs. 5.32 and 5.33 is a small picture describing various screen coordinates that define different “action zones” for the robot. The robot’s goal is to keep the barycenter of the Color Object within the box defined by the four **object_WX** parameters.

Fig. 5.32 lists the first and second sets of event-action pairs used to maneuver the robot into keeping the Color Object centered and with a given pixel area size:

- Line 252 shows that the CM-550 will only use the Color Object data when it is not moving, i.e. when (**motion.status() == FALSE**).

```

251 # Autonomous response to object type packets received
252 if ((object_Area > 0) and (object_X > 0) and (object_Y > 0) and
253     # tilt adjustments (VSR_Control == True) and (motion.status() == False)):
254     if ((object_Y < res_Height) and (object_Y >= object_WD)):
255         down_tilt()
256     elif ((object_Y <= object_WU) and (object_Y > 0)):
257         up_tilt()
258     elif ((object_Y > object_WU) and (object_Y < object_WD)):
259         etc.write8(199,3)
260         while (etc.read8(199) != 0):
261             delay(0)
262 # slight sideways adjustments to Quadruped
263 if ((object_X > x1) and (object_X <= object_WL)):
264     left_slide()
265 elif ((object_X >= object_WR) and (object_X < x3)):
266     right_slide()

```

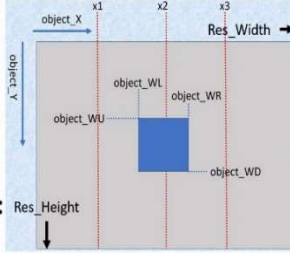


Fig. 5.32 Part 1 of Handling Procedures for Color Object data in “QUAD_DRC_VSR_PC_RPi.py”.

- Lines 254-255 show that when the Color Object is found below the **object_WD** horizontal line, the robot ought to tilt down its body/camera.
- Lines 256-257 show that when the Color Object is found above the **object_WU** horizontal line, the robot ought to tilt up its body/camera.
- Lines 258-261 show that when the Color Object is found within the “blue box”, the robot ought to clear all its offsets to zero (i.e. send a **3** to Address 199 – Line 259) and then wait a bit for this process to finish (Lines 260-261).
- Lines 263-264 show that when the Color Object is found within the two vertical lines defined by **x1** and **object_WL**, the robot ought to “slide left”.
- Lines 265-266 show that when the Color Object is found within the two vertical lines defined by **x3** and **object_WR**, the robot ought to “slide right”.

Fig. 5.33 lists the third set of event-action pairs used to maneuver the robot into keeping the Color Object centered and with a given pixel area size:

- Lines 269-270 show that when the Color Object is found to the left of vertical line **x1**, the robot ought to “turn left”.
- Lines 271-272 show that when the Color Object is found to the right of vertical line **x3**, the robot ought to “turn right”.
- Lines 273-274 show that when the Color Object size is found to be within its Low and High limits, the robot ought to “stop”.
- Lines 275-276 show that when the Color Object size is found to be larger than its High limit, the robot ought to “go backward”.

- Lines 277-278 show that when the Color Object size is found to be smaller than its Low limit, the robot ought to “go forward”.

```

268         # larger adjustments to Quadruped
269         if ((object_X > 0) and (object_X <= x1)):
270             turn_left()
271         elif ((object_X >= x3) and (object_X <= res_Width)):
272             turn_right()
273         elif ((object_Area >= object_Area_Low) and (object_Area <= object_Area_High)):
274             stop()
275         elif ((object_Area > object_Area_High)):
276             go_backward()
277         elif ((object_Area < object_Area_Low)):
278             go_forward()

```

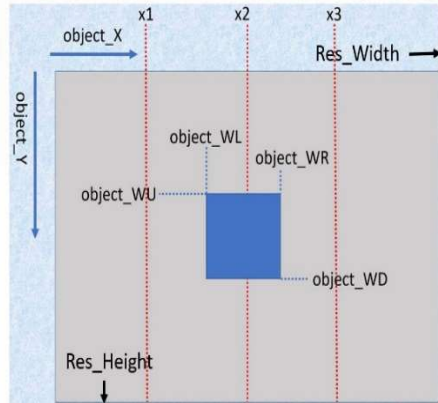


Fig. 5.33 Part 2 of Handling Procedures for Color Object data in “QUAD_DRC_VSR_PC_RPi.py”.

Please note that for each iteration of the Main Endless Loop, the robot could trigger from none to all three of the three sets of event-action pairs listed in Figs. 5.32 and 5.33, because 3 separate IF-ELSE-IF structures were used.

The solution for the Obstacle Avoidance (SA for Smart Avider) project is named “QUAD_SA_DRC_VSR_PC_RPi.py” and only the main programming features will be further elaborated on in this Sub-Section.

Fig. 5.34 describes the new variables, objects and functions needed to implement the SA feature:

- The new variables are **threshold_1**, **threshold_2**, **threshold_3**, **obstacle**, and **nir_difference**.
- The two new object instances are **nir_left** and **nir_right** (Lines 177 and 178) – for DMS-80s.
- Three new functions were also needed:
 - **reg_stop()**: to play **Motion 0**, i.e. to stop the robot after playing of current Motion to its end.
 - **emer_stop()**: to play **Motion -3**, i.e. to stop the robot “immediately”, at whatever pose it happens to be it at the time **Motion -3** is invoked.
 - **success()**: to call **reg_stop()**, play the buzzer and reset Parameter **obstacle** back to FALSE.

```

173 threshold_1 = 550
174 threshold_2 = 475
175 threshold_3 = 400
176 obstacle = False
177 nir_left = OLLO(3, const.OLLO_DMS)
178 nir_right = OLLO(4, const.OLLO_DMS)
179 nir_difference = 0

150 def reg_stop():
151     motion.play(0)
152     delay(1000)
153
154 def emer_stop():
155     motion.play(-3)
156     delay(1000)
157
158 def success():
159     global obstacle
160     reg_stop()
161     buzzer.melody(3)
162     buzzer.wait()
163     delay(2000)
164     obstacle = False

```

Fig. 5.34 New Variables, Objects and Functions needed for implementing SA feature in “QUAD_SA_DRC_VSR_PC_RPi.py”.

```

202 # If no obstacle in front, OK to follow user's RC instructions
203 if ((nir_left.read() < threshold_1) and (nir_right.read() < threshold_1)):
204     data_in = 0
205     comm_Type = -1
206     if (rc.received() == True):

321 else:
322     # Obstacle is getting close - Robot stops first for 1 second then tries to avoid obsta
323     obstacle = True
324     # reg_stop()
325     emer_stop()
326     # delay(1000)
327     while ((nir_left.read() > threshold_2) or (nir_right.read() > threshold_2)):
328         nir_difference = nir_left.read() - nir_right.read()
329         if ((nir_difference <= 10) and (nir_difference >= -10)):
330             go_backward()
331             while ((nir_left.read() > threshold_3) and (nir_right.read() < nir_left.read())):
332                 turn_right()
333             while ((nir_right.read() > threshold_3) and (nir_left.read() < nir_right.read())):
334                 turn_left()
335
336     # Here, no more obstacle in front - Robot stops and plays music to celebrate avoidance
337     success()
338     # Checking if operator has released all RC buttons
339     while ((rc.received() == True)): # compact format for read & check on rc packets rece:
340         delay(1)

```

Fig. 5.35 New Procedure for implementing SA feature in “QUAD_SA_DRC_VSR_PC_RPi.py”.

Fig. 5.35 shows that the Main Endless Loop was modified to accommodate the SA feature:

- Line 203 shows that if both NIR Sensors read in values less than **threshold_1**, that would be considered as “no obstacle is found nearby”, then the procedures for processing Remocon packets of **comm_Type** 0 and 1 are activated as before for “QUAD_DRC_VSR_PC_RPi.py”.
- When either of the NIR Sensors read in a value larger than **threshold_1**, that would signify that “some obstacle is found nearby”, the ELSE branch (Lines 321-340) will be executed next.
- Parameter **obstacle** is set to TRUE (Line 323), then the user has the choice of using **reg_stop()** (Line 324) or **emer_stop()** (Line 325). A time delay of 1 sec. is optional to use (Line 326).
- The OUTER WHILE LOOP (Lines 327-334) is used to figure out the exact location of the obstacle and perform the appropriate robot maneuvers to steer it clear (enough) of the obstacle:
 - Line 327 indicates that when either NIR Sensor registers a value bigger than **threshold_2**, this would mean that the obstacle is “still near” the robot, therefore the robot needs to figure out and perform the appropriate maneuvers to get it away from the obstacle.
 - Lines 328-330 are used to figure out if the obstacle is located right in front of the robot, if it is so, it needs to go backward (Line 330).
 - The first INNER WHILE LOOP (Lines 331-332) is used to figure out if the obstacle is located to the left of the robot, if it is so, it needs to turn right (Line 332).
 - The second INNER WHILE LOOP (Lines 333-334) is used to figure out if the obstacle is located to the right of the robot, if it is so, it needs to turn left (Line 334).
- When the OUTER WHILE LOOP is exited, meaning that the NIR Sensors register “all clear”, Function **success()** (Line 337) is called to stop the robot, plays the buzzer and resets Parameter **obstacle** to FALSE.
- The last WHILE LOOP (Lines 339-340) is used to wait on the operator to release all keys on the keyboard attached to the RPi4B.

Ideas for further explorations (HFE 5.1):

5.2.1: *The reader may remove the condition (**motion.status()** == **False**) from Line 252 in Fig 5.32 to see if the E-QUAD runtime performance would improve? Or maybe it would get worse?*

5.2.2 Using Time Control and Motion Arrays

For the XL430 series, the Time Control option needs first to be set in the EEPROM Parameter “Drive Mode” (Address 10 and Bit 2 = 1, please review beginning of Section 2.2 of Thai (2020-a) if needed). When in a Time-based Position Control mode, the parameters “Profile Velocity” (Address 112) and “Profile Acceleration” (Address 108) are still used to set up the “Velocity Trajectory” (VT) and “Position Trajectory” (PT), but they will have a completely different meaning: **their numerical values represent “milliseconds” in a TIMED Position Control mode.**

The link <http://emanual.robotis.com/docs/en/dxl/x/2xl430-w250/#profile-velocity112> has some information about the TIMED Position Control (PC) option, but it is not complete. In actuality, the TIMED PC option can accommodate a RECTANGULAR or a TRAPEZOIDAL profile in a similar way as for the “regular” PC Mode 2 and 3 (please review Section 2.2 of Thai (2020-a) if needed).

5.3 Using Python on RPi4B/Desktop PC

This Section 5.3 is common to TASK and MicroPython coders and it uses Standard Python. It should be read after either Section 5.1 or 5.2 had been studied. The two projects described in this section implement the overall communications scheme developed for E-PTC in Fig. 4.2 but here adapted for E-QUAD and its variants:

1. The first project is developed for the RPi4B, equipped with the Pi Camera, to work as Vision Processor and Remote Controller for the E-QUAD robot using the keyboard from the Desktop PC via VNC Viewer into the RPi4B.
2. The second project is designed to allow the Desktop PC to work as a Central Data Hub receiving various sensors and actuators data from the E-QUAD and its variants.

5.3.1 RPi4B as Vision Processor/Remote Controller

The Standard Python solution to this project is named “QUAD_RC_Color_Tracker_XY_RPi.py” and it is virtually identical to the E-PTC’s solution named “PTC_RCSD_Color_Tracker_XY_RPi.py” (see Sub-Section 4.3.1), except for the use of the keyboard (see Fig. 5.55). Thus, the materials provided in Sub-Section 4.3.1 will not be repeated here – please review if needed.

```
74 message1 = "Use keyboard to perform various operations:"
75 message2 = "ESC to quit program"
76 message3 = "!!! Push on s or S for Emergency Stop of Robot"
77 message4 = "U-D-L-R Arrows to move Robot / 1-3 for Body Tilting / 2-4 for Sliding Left or Right"
78 message5 = "5 to increase Robot's Speed / 6 to decrease Robot's Speed"
79 message6 = "E or e for EXTRACTION of COLOR data for NEW OBJECT"
80 message7 = "T or t to track for CHOSEN OBJECT and put Robot into VSR actions"
81 message8 = "\t Once in TRACK Mode, press R or r to quit Robot's VSR actions"
```

Fig. 5.55 Keyboard Usage in “QUAD_RC_Color_Tracker_XY_RPi.py”.

The program “QUAD_RC_Color_Tracker_XY_RPi.py” is designed to work with the previously developed MicroPython codes “QUAD_DRC_VSR_PC_RPi.py” or “QUAD_SA_DRC_VSR_PC_RPi.py” (Sub-Section 5.2.1) or “TC_QUAD_SA_DRC_VSR_PC_RPi.py” (Sub-Section 5.2.2).

At runtime, the reader certainly would notice that “QUAD_RC_Color_Tracker_XY_RPi.py” does not perform as well as “PTC_RCSD_Color_Tracker_XY_RPi.py”. The main reason is that a

“walking” robot such as E-QUAD does not provide a very stable platform for tracking a moving object as compared to a wheel-based robot such as the E-PTC. Furthermore, Fig. 5.32 shows that all 3 CM-550 side codes use Object Tracking Data, sent over from the RPi4B, **only when the E-QUAD is not moving** (i.e. `motion.status() == False`). This means that the E-QUAD robot receives information about its “environment” **much less often** than the E-PTC robot, and this slower information flow undoubtedly induces a slower reaction rate for E-QUAD. For this reason, the author designed the A4WP-H variant as a mixture of “walking” and “rolling” control for use in Sections 5.4 and 5.5 as an effort to improve object tracking performance with “walking” robots.

5.3.2 Desktop PC as Central Data Hub

In this Sub-Section 5.3.2, a self-standing Python program emulates the Output Monitor functionality of the ROBOTIS TASK tool, and its name is “QUAD_Data_Central_PC.py”. It is virtually identical to the E-PTC’s solution named “PTC_Data_Central_PC.py” (see Sub-Section 4.3.2), except for the use of different key presses and robot images (see Fig. 5.56). Thus, the materials provided in Sub-Section 4.3.2 will not be repeated here – please review if needed.

```

36 message1 = "Use keyboard to perform various operations:"
37 message2 = "ESC to quit program"
38 message3 = "Toggle Key Q ON/OFF for Quadruped's data flow" ←
39
40 # OpenCV section
41 im_QUAD_ON = "Quadruped_On_S.jpg" # 288x321 pixels ←
42 im_QUAD_OFF = "Quadruped_Off_S.jpg" ←
43 window_1 = "Quadruped"
44 robot_1 = False
45 received_line_1 = 0

```

Fig. 5.56 Keyboard and Images Usage in “QUAD_Data_Central_PC.py”.

5.4 Using C++ on RPi4B and Desktop PC

This Section 5.4 is common to TASK and MicroPython coders and it uses Standard C++. It should be read after either Section 5.1 or 5.2 had been studied. The two projects described in this section implement the overall communications scheme developed for E-PTC in Fig. 4.2 but here adapted for A4WP-H:

1. The first project is developed for the RPi4B, equipped with the Pi Camera, to work as Vision Processor and Remote Controller for the A4WP-H robot using the keyboard from the Desktop PC via VNC Viewer into the RPi4B.
2. The second project is designed to allow the Desktop PC to work as a Central Data Hub receiving various sensors and actuators data from the A4WP-H.

On the CM-550 side, the codes “A4WP-H_DRC_VSR_PC_RPi.tsk3/mtn3” (Sub-Section 5.1.4) or “A4WP-H_DRC_VSR_PC_RPi.py” (Sub-Section 5.2.2) can be used with these two projects.

5.4.1 RPi4B as Vision Processor/Remote Controller

The solution for this A4WP-H project is named “A4WP-H_RC_Color_Tracker_RPi.cpp” and it is virtually identical to its E-PTC equivalent “PTC_RC_Color_Tracker_XY_RPi_BST.cpp” (see Sub-Section 4.4.1). The only difference is only in the way that certain key presses will now point to difference robot actions (see Fig. 5.57).

```
248 // Print out simple menu
249 cout << "Use keyboard to perform various operations:" << endl;
250 cout << "ESC to quit program" << endl;
251 cout << "!!! Push on s or S for Emergency Stop of Robot" << endl;
252 cout << "U-D-L-R Arrows to move Robot / 1-3 for Body Tilt / 4 for Robot Reset" << endl;
253 cout << "5 to increase Robot Speed / 6 to decrease Robot Speed" << endl;
254 cout << "E or e for EXTRACTION of COLOR data for NEW OBJECT" << endl;
255 cout << "T or t to track for CHOSEN OBJECT and put Robot into VSR actions" << endl;
256 cout << "\t Once in SCAN Mode, press R or r to quit Robot's VSR actions" << endl;
```

Fig. 5.57 Keyboard Usage in “A4WP-H_RC_Color_Tracker_RPi.cpp”.

5.4.2 Desktop PC as Central Data Hub

The solution for this A4WP-H project is named “A4WP-H_Data_Central_PC.cpp” and it is virtually identical to its E-PTC equivalent “PTC_Data_Central_PC.cpp” (see Sub-Section 4.4.2). The only difference is only in the images and key presses that are now used (see Fig. 5.58).

```
56 // OpenCV Init Section
57 Mat img_1, img_2, img_3;
58 key_1 = 0;
59 key_2 = 0;
60 const char* im_A4WP_ON = "A4WP-H_On_S.jpg"; / 302x302 pixels
61 const char* im_A4WP_OFF = "A4WP-H_Off_S.jpg";
62 const char* window_1 = "A4WP-H";
63
64 // Create Windows
65 img_1 = imread(im_A4WP_OFF);
66 imshow(window_1, img_1);
67 moveWindow(window_1, 0, 0);
68
69 // Print out simple menu
70 cout << "Use keyboard to perform various operations:" << endl;
71 cout << "ESC to quit program" << endl;
72 cout << "Toggle Key (a) ON/OFF for A4WP-H's data flow" << endl;
```

Fig. 5.58 Keyboard and Images Usage in “A4WP-H_Data_Central_PC.cpp”.

5.5 Applying IDN Concept to A4WP-H with DXL-HAT + DXL-SDK

When the author applied the Independent Dynamixel Network concept to the A4WP-H robot, interesting mechanical and communications issues arose. From a systems view of the A4WP-H robot, there is no “Split” DXL Control option like for the E-PTC, thus all 12 XL430 actuators would have to be under the control of the RPi4B, leaving the CM-550 to manage its local sensors. Additionally, the CM-550 needs to communicate to the RPi4B about the status of those sensors so that the RPi4B can activate appropriate robot maneuvers at runtime.

Thus, for this Section 5.5, the goal is to use the A4WP-H frame to create an Obstacle Avoiding robot that has all its servos controlled by the RPi4B via the DXL-HAT and using Time-Control mode, while the CM-550 manages the DMS-80 and IMU sensors.

The C++ solution, running on the RPi4B, is named “IDN_A4WP-H_SA_RC_VSR_RPi.cpp”, and the corresponding CM-550 solutions are named “IDN_A4WP-H_SA_RC_RPi.task3/py”.

5.5.1 Mechanical Cabling Issue

First, the IDN version of the A4WP-H robot would need its X3P DXL cable system redesigned so that the CM-550 powers all the XL430s but cannot see/control them. The author’s first design iteration resulted in a solution that required long X3P cables between the robot legs (see Fig. 5.59). The robot had no problems performing its required maneuvers, but these cables tended to snag on things off the running surface.

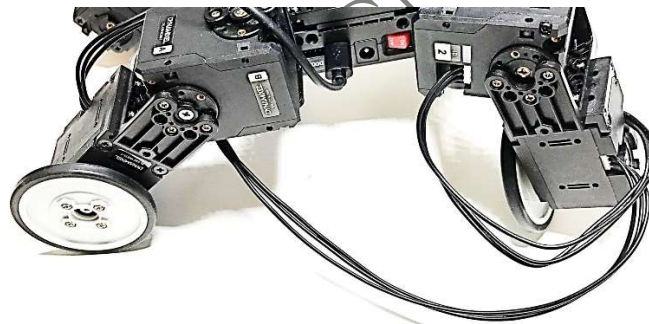


Fig. 5.59 First IDN Cabling Solution for A4WP-H robot.

Fig. 5.61 illustrates how the last “regular” X3P cable, connecting one robot leg to the JST bus on the DXL-HAT, allows the RPi4B/DXL-HAT to “see” all the XL430s and thus can control them via the DXL-SDK and through the RPi4B.

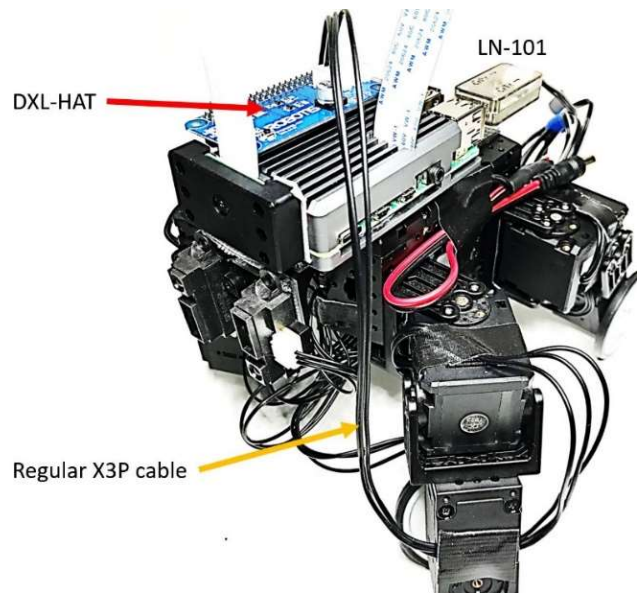


Fig. 5.61 Controlling XL430s with DXL-HAT/RPi4B for A4WP-H robot.

5.5.2 CM-550 to RPi4B Communications Issue

In all the RPi4B projects so far, the author had been able to use Port **ttyACM0** to send Remocon packets from the RPi4B to the CM-550. But for this current project, a reverse data flow would be needed, as the CM-550 needs to send either raw DMS-80 sensor data or appropriate directives to the RPi4B to tell it about the presence of an obstacle in the robot path. The RPi4B can then issue appropriate maneuver commands to the XL430s so that the A4WP-H robot can clear the obstacle.

Furthermore, the author wanted to use the ZGB-SDK for this reverse data flow to see if it can coexist with the DXL-SDK within a C++ environment on the RPi4B. So, he recompiled the ZGB-SDK to use Port **ttyACM0** and incorporated it into the C++ solution for this project which then compiled fine. However, at runtime, this compiled code could not connect to Port **ttyACM0** for some reasons. But if the author went back to using an LN-101, connecting an RPi4B's USB Port (i.e., **ttyUSB0**) to the UART Port on the CM-550, the ZGB-SDK connection worked fine between RPi4B and CM-550! Thus, it looks like that ROBOTIS Software work best with ROBOTIS Hardware!

The RPi4B main task is to manage the Pi Camera and the Time-Controlled XL430s via DXL-SDK and Port **ttyS0** where the DXL-HAT is connected to. The Bluetooth **rfcomm0** device on RPi4B acts as the wireless communications link between PC and RPi4B. As previously mentioned in Section 4.6, **rfcomm0** only works with a Python/PySerial code running on the PC.

5.5.3 CM-550's TASK & MicroPython Solutions

Let us next look at the CM-550 solutions "IDN_A4WP-H_SA_RC_RPi.tsk3/py". Both versions implemented the same "logic", but the author will use the MicroPython version to explain the various procedures used. The corresponding C++ code on the RPi4B is named "IDN_A4WP-H_SA_RC_VSR_RPi.cpp". Fig. 5.63 lists 4 Functions used in "IDN_A4WP-H_SA_RC_RPi.py":

- Function **alarm()** plays the musical note 3 for 0.5 second (Lines 12-13).
- Function **init_com_VSR()** sets the **use mode** for the three serial communication ports on the CM-550 (please review Fig. 5.62 also):
 - The **UART** Port (Line 16) is set to “Remote Port” where Remocon packets will be received or transmitted from. This **UART** Port is connected to Port **tttyUSB0** on the RPi4B via an LN-101.
 - The **App** Port is not used in this project, so it is just set to its default use (BLE) (Line 17).
 - The **Task Print** Port (Line 18) is set to **BLE** which is the Embedded BT-410 on the CM-550, meaning that the PC must be using a BT-410 USB dongle to connect to the Embedded BT-410 on the CM-550.

5.5.4 RPi4B's C++ Solution

Fig. 5.71 shows a simple menu to indicate the various “robot actions” that are implemented in the program “IDN_A4WP-H_SA_RC_VSR_RPi.cpp”. Please note that Keys 3 and 4 are not used.

```

1030 // Print out simple menu
1031 cout << "Use keyboard to perform various operations:" << endl;
1032 cout << "ESC to quit program" << endl;
1033 cout << "!!! Push on s for Emergency Stop/Freeze of Robot" << endl;
1034 cout << "!! Push on r for Reset to Robot Init Pose" << endl;
1035 cout << "U-D-L-R Arrows to move Robot / 1 for Robot Tilt Up / 2 for Robot Tilt Down" << endl;
1036 cout << "5 to increase Robot Speed / 6 to decrease Robot Speed" << endl;
1037 cout << "E or e for EXTRACTION of COLOR data for NEW OBJECT" << endl;
1038 cout << "T or t to track for CHOSEN OBJECT and put Robot into VSR actions" << endl;
1039 cout << "----- Press any key to continue" << endl;
1040 getchar();

```

Fig. 5.71 Simple menu used in “IDN_A4WP-H_SA_RC_VSR_RPi.cpp”.


```

23  #include "dynamixel_sdk.h"
24  #include <zigbee.h>
25
26  #define DEFAULT_DEVICEINDEX 0 // /dev/ttyUSB0 for ZGB-SDK
27  #define TIMEOUT_TIME 50 // msec
28
29  #define BAUDRATE 1000000
30  #define DEVICENAME "/dev/ttyS0"
31
32  int IDs_GP[8] = {1,2,3,4,5,6,7,8}; // IDs of DXL under Position Control
33  int IDs_GV[4] = {11,12,13,14}; // IDs of DXL under Velocity Control
34  uint8_t dxl_error = 0; // Dynamixel error
35  uint8_t moving_flag = 0; // Individual DXL's moving flag
36  uint8_t all_moving_flags = 0; // Sum of all moving flags
37  uint8_t param_goal_position[4]; // byte array for goal position value
38  uint8_t param_goal_velocity[4]; // byte array for goal velocity value
39  int32_t dxl_present_position[8] = { 0 }; // Present position array
40  bool read_present_positions_OK = false;
41  bool wheels_only = false;

```

Fig. 5.72 Selected C++ Definitions used in “IDN_A4WP-H_SA_RC_VSR_RPi.cpp”.

Fig. 5.72 shows selected C++ Definitions used in “IDN_A4WP-H_SA_RC_VSR_RPi.cpp”:

- This program is using the DXL-SDK (Line 23) and the ZGB-SDK (Line 24), and that Port **tttyUSB0** is used with the ZGB-SDK (Lines 26-27), while Port **tttyS0** is used with the DXL-SDK (Lines 52-53). The reader/user may have to change the ownership rights on **tttyUSB0/tttyS0** once (before running any executable program, using a **bash** Terminal to issue the following commands:
 - **sudo chmod a+rw /dev/tttyACM0**
 - **sudo chmod a+rw /dev/tttyUSB0**
- Lines 64-65 list the IDs arrays used in many Functions that set Goal Position or Goal Velocity commands to the robot’s XL430 actuators.
- Lines 66-71 lists specific data types that are used by the DXL-SDK (coming from ROBOTIS example source codes).
- Lines 72-73 shows some other author-created variables used in this program.

5.6 Supervisory Control of A4WP-H (and E-PTC)

In this last project for Chapter 5, the goal is to set the Desktop PC as a Supervisory Controller, meaning that the PC can override the RPi4B’s current commands to the CM-550 (see Fig. 5.62).

The RPi4B would now run on “IDN_A4WP-H_SA_SC_VSR_RPi.cpp” which is modified from the program “IDN_A4WP-H_SA_RC_VSR_RPi.cpp” (described in Sub-Sections 5.5.3 and 5.5.4) to incorporate a new BT communication channel via **rfcomm0** connecting between the RPi4B and the PC. Please refer to Appendix B, Sub-Section B.3, for more details about setting/using **rfcomm0**.

As previously discussed in Section 4.6, the companion PC solution needs to be a Python program, and it is named “SC_PTC_A4WP_2BT_Central_BST.py”. This PC Python program is designed to serve as an interface to both E-PTC and A4WP-H robots.

5.6.1 RPi4B C++ Solution for A4WP-H

All functionalities implemented in “IDN_A4WP-H_SA_RC_VSR_RPi.cpp” are retained in “IDN_A4WP-H_SA_SC_VSR_RPi.cpp” and the only new programming features relate to how to receive and implement the commands from the PC sent to the RPi4B but meant for the CM-550. The pertinent programming steps in “IDN_A4WP-H_SA_SC_VSR_RPi.cpp” are described in Figs. 5.93 to 5.99.

```
25 | #include "dynamixel_sdk.h" ←
26 | #include <zigbee.h> ←
27 |
28 | #define DEFAULT_DEVICEINDEX 0 // /dev/ttyUSB0 for ZGB-SDK
29 | #define TIMEOUT_TIME 50 // msec
30 |
31 | using namespace std;
32 | using namespace cv;
33 | using namespace ::boost::asio; ←
```

Fig. 5.93 Selected include packages and definitions used in “IDN_A4WP-H_SA_SC_VSR_RPi.cpp”.

5.6.2 Desktop PC Python Solution for A4WP-H and E-PTC

The companion PC solution “SC_PTC_A4WP_2BT_Central_BST.py” was already described once in Section 4.6, but only for the E-PTC robot. Figs. 5.100 to 5.103 will describe the additional features to accommodate both E-PTC and A4WP-H robots.

Fig. 5.100 lists selected Initializations and Definitions needed at the beginning of the Python code for “SC_PTC_A4WP_2BT_Central_BST.py”:

- On the author’s PC setup, Line 109 shows that the BT connection to the E-PTC robot (**btser1**) is done via **COM14**, while the BT connection to the A4WP-H robot (**btser2**) is done via **COM20** (Line 113).
- Lines 144-148 provide a simple menu for the Operator to use this program.
- Lines 151-156 list various graphics objects used in this project.

```

109 btser1 = serial.Serial("COM14", 115200, timeout=0, write_timeout=0) #
110 btser1.reset_input_buffer() # clear buffer for data coming in from Rf
111 btser1.reset_output_buffer() # clear buffer for data sent to RPi-PTC
112
113 btser2 = serial.Serial("COM20", 115200, timeout=0, write_timeout=0) #
114 btser2.reset_input_buffer() # clear buffer for data coming in from Rf
115 btser2.reset_output_buffer() # clear buffer for data sent to RPi-A4WP

144 message1 = "Use keyboard to perform various operations:"
145 message2 = "ESC to quit program"
146 message3 = "Toggle Key p for PTC or Key a for A4WP"
147 message4 = "Use UDLR123456 keys to control the robot(s) - only 1-key push allowed"
148 message5 = "!!! Push on s for Emergency Stop"
149
150 # OpenCV section
151 im_PTC_ON = "E-PTC_RPi_On_S.jpg" # 484x484 pixels
152 im_PTC_OFF = "E-PTC_RPi_Off_S.jpg"
153 im_A4WP_ON = "A4WP-H_On_S.jpg" # 302x302 pixels
154 im_A4WP_OFF = "A4WP-H_Off_S.jpg"
155 window_1 = "E-PTC"
156 window_2 = "A4WP-H"

```

Fig. 5.100 Selected Initializations and Definitions implemented in “SC_PTC_A4WP_2BT_Central_BST.py”.

5.6.3 Supervisory Control Demonstration for A4WP-H

Fig. 5.104 shows the overall mechanical and communications configurations of the A4WP-H robot in its IDN version.

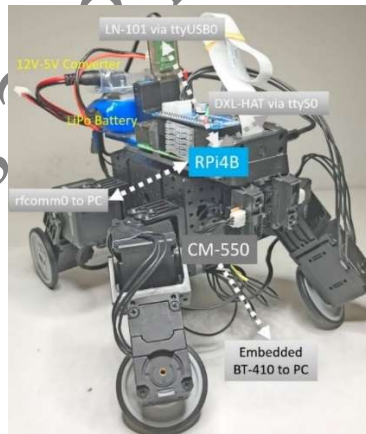


Fig. 5.104 Mechanical and Communications Configurations for A4WP-H (IDN version).

In summary for this A4WP-H Supervisory Control project, there are 4 programs that need to be executed in a coordinated fashion:

- On the PC, “SC_PTC_A4WP_2BT_Central_BST.py” and “A4WP-H_Data_Central_PC.cpp”.

- On the RPi4B, “IDN_A4WP-H_SA_SC_VSR_RPi.cpp”.
- On the CM-550, “IDN_A4WP-H_SA_RC_RPi.tsk3/py”.

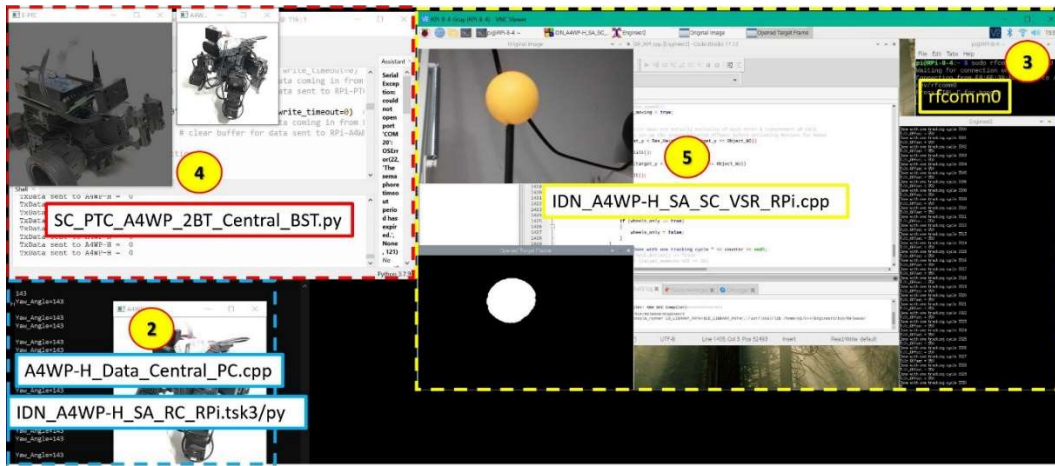


Fig. 5.105 Runtime Screen Capture for project “PC as Supervisory Controller” for the A4WP-H robot.

5.6.4 Supervisory Control Demonstration for A4WP-H + E-PTC

As the final demonstration for Chapter 5, we’ll use the PC to control both E-PTC and A4WP-H robots. Both 5-step procedures shown in Sub-Sections 4.6.2 and 5.6.3 must be combined to synchronize 1 PC, 2 RPi4Bs and 2 CM-550s for them to work together (see Fig. 5.106):

- In the top-left of Fig. 5.106 is a window for the Thonny IDE running the program “SC_PTC_A4WP_2BT_Central_BST.py”.
- At the bottom-left in Fig. 5.106 are two “Data Flow” windows, one for “A4WP-H_Data_Central_PC.cpp” and the other for “E-PTC_Data_Central_PC.cpp”.
- In the top-middle area is an external view via Web Cam of the overall physical set up, showing that the A4WP-H robot was tracking a green ball, while the E-PTC robot was tracking a blue ball.
- In the right area of Fig. 5.106 are two VNCViewer windows displaying the desktops of the E-PTC/RPi4B and A4WP-H/RPi4B interfaces.

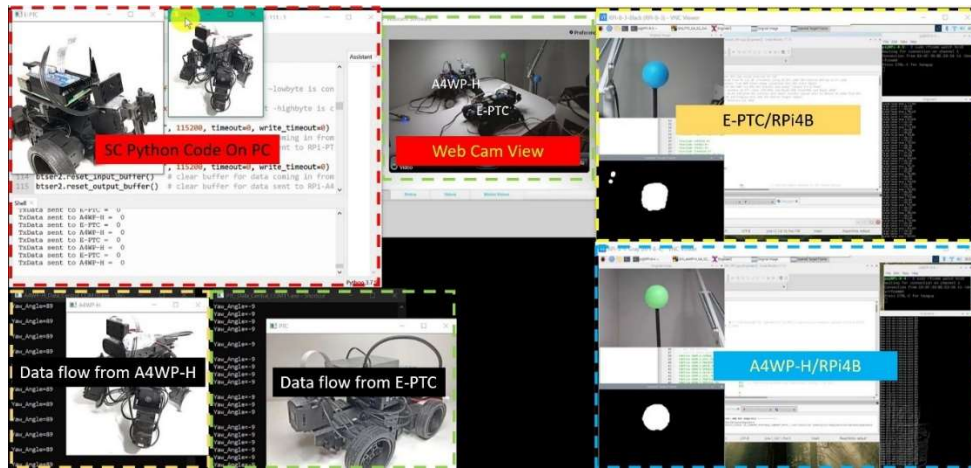


Fig. 5.106 Runtime Screen Capture for project “PC as Supervisory Controller” for E-PTC and A4WP-H robots.

A narrated video of about 5 minutes of runtime demonstration of this project is available on YouTube (<https://www.youtube.com/watch?v=w6YSA-1m07s>). Both robots behaved as programmed during the first 3.5 minutes, but afterwards both RPi4Bs started to lock up, then the A4WP-H/RPi4B stopped its VNC Server, thus there was a total loss of control over the A4WP-H robot after that event. So, the RPi4Bs were not quite robust as the author had hoped for, perhaps because Raspian 64-bit OS is still in its beta phase currently (September 2021).

Chapter 6: Enhanced SPI: CM-550 vs. RPi4B vs. Jetson Nano

Fig. 6.1 shows the author's mechanical modifications to the original design for the SPI robot which came with the ENGINEER Kit 1:

1. Two additional XL430-W250-T are used as Servos 15 and 16, and one Frame Part EF25-F23 connects them together to work as a Pan-Tilt platform for the Pi Camera.
2. Two DMS-80s are also mounted in front of the Frame Part EF25-F23 to serve as obstacle sensors.

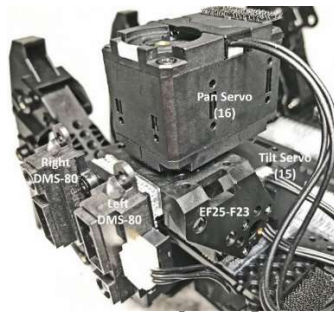


Fig. 6.1 Additional Pan-Tilt Platform and DMS-80s for the Original SPI Robot.

This Enhanced SPI (E-SPI) robot is designed to carry either an RPi4B or a Jetson Nano 2GB on its top surface, therefore the CM-550 and LiPo battery get shifted to the robot's bottom area (see Fig. 6.2).



Fig. 6.2 CM-550 and LiPo Battery shifted to the bottom area of the E-SPI Robot.

The projects for the E-SPI robot have similar goals to the previous robots E-PTC and A4WP-H robot:

- Mixing Autonomous and Remote Controls schemes.

- Use of the SBCs (RPi4B or Jetson Nano) as the Machine Vision Processor.
- Applications of the Independent Dynamixel Networks (IDN) approach to best distribute the computing load between the SBC and CM-550.

Based on results obtained from Chapters 4 and 5, the author decided that the Pan-Tilt Platform (Servos 15 and 16) and the Pi Camera would be controlled by the SBC in use, while the Multipedal Platform and the DMS-80s would be controlled by the CM-550.

Three E-SPI projects are presented in this Chapter 6:

1. A camera-less project using only the CM-550 that can be served as a foundation for the next two SBC-based projects.
2. A Machine Vision based project using the RPi4B.
3. Another Machine Vision based project using the Jetson Nano.

6.1 Camera-less Solution in TASK/MicroPython

In this project, all 12 servos of the Multipedal Platform and the 2 servos of the Pan-Tilt Platform are controlled by the CM-550 (see Fig. 6.3):

- The Multipedal Platform's movements are based on the Motion Units provided in the ROBOTIS example SPI MTN3 file but modified to suit the author's needs.
- The Pan-Tilt platform is controlled via SyncWrite Goal Position commands.

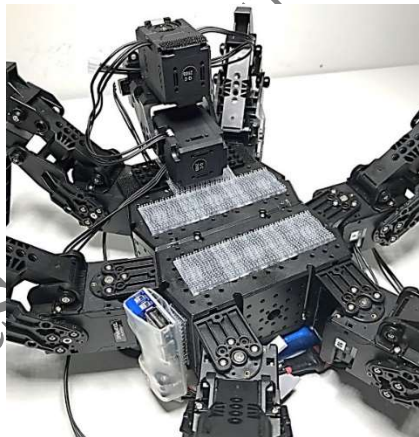


Fig. 6.3 Camera-less E-SPI project using only CM-550.

6.2 Machine Vision Solution with RPi4B

In this Section 6.2, a Machine Vision solution using the RPi4B is documented:

- The solutions for the CM-550 were created in TASK and MicroPython (see enclosed source codes), however for “variety” sake, only the MicroPython version “IDN_SPI_SA_RC_RPi.py” will be described further in this Section 6.2.
- For the RPi4B, Chapters 4 and 5 already showed that a C++ approach was needed to provide the best runtime performance out of the RPi4B, thus the author did not create a

Python version of the RPi4B C++ solution which is named “IDN_SPI_SA_SSC_VSR_RPi.cpp”.

6.2.1 Hardware/Communication Configurations for RPi4B + E-SPI

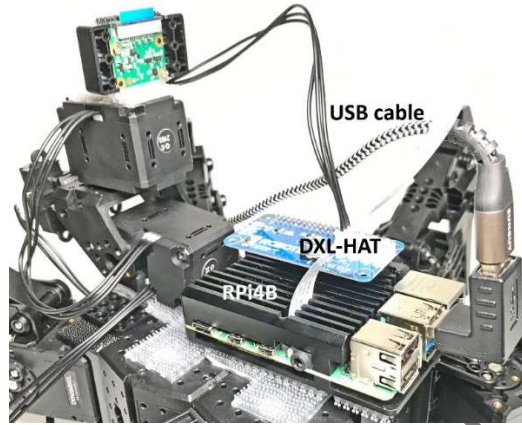


Fig. 6.12 Hardware Configuration used for integrating RPi4B to CM-550.

Fig. 6.12 shows the Hardware Configuration used for this project:

- An RPi4B with a compact passive cooling system is used.
- A DXL-HAT is used to allow the RPi4B to control the Pan-Tilt platform and Pi Camera directly via the DXL-SDK.
- A plain USB cable is used to connect the RPi4B to the CM-550 to allow one-way Remote communications between RPi4B and CM-550. Thus, the RPi4B only issues high-level movement commands “UDLR12456” to the CM-550 which takes care of the low-level Motion Units/Lists aspects to move the E-SPY multipedal chassis, while avoiding potential obstacles via the DMS-80 sensors.

6.3 Machine Vision Solution with Jetson Nano 2GB

In this Section 6.3, a Machine Vision solution using the Jetson Nano 2 GB is documented:

- The solutions for the CM-550 were created in TASK and MicroPython (see enclosed source codes), but only the MicroPython version “IDN_SPI_SA_RC_JN.py” will be described further in this Section 6.3.
- As Chapters 4 and 5 already showed that a C++ approach was needed to provide the best runtime performance out of SBCs, the author did not create a Python version of the Jetson Nano’s C++ solution which is named “IDN_SPI_SA_SSC_VSR_JN.cpp”.

6.3.1 Hardware/Communication Configurations for J-Nano + E-SPI

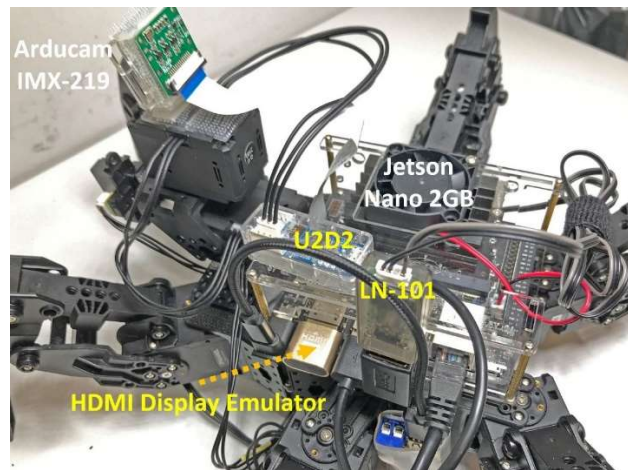


Fig. 6.27 Hardware Configuration used for integrating Jetson Nano to CM-550.

Fig. 6.27 shows the Hardware Configuration used for this project:

- A standard Jetson Nano 2GB with a 4 GB of swap space is used (please see Appendix C for more details).
- A U2D2 module is used to allow the J-Nano to control the Pan-Tilt platform via DXL-SDK and an IMX-219 based Camera directly. Some folks had used the DXL-HAT (designed for RPi only) with J-Nano successfully, but at this point (August 2021), ROBOTIS does not fully recommend the use of the DXL-HAT with J-Nano yet.
- The author could have used a plain USB cable to connect the J-Nano to the CM-550 to allow Remocon communications between J-Nano and CM-550. But, for “variety” sake, he chose the LN-101 instead. Thus, similarly to the RPi4B, the J-Nano only issues high-level movement commands “UDLR12456” to the CM-550 which takes care of the low-level Motion Units/Lists aspects to move the E-SPY multipedal chassis, while avoiding potential obstacles via the DMS-80 sensors.

6.3.3 J-Nano Solution “IDN_SPI_SA_SSC_VSR_JN.cpp”

The J-Nano solution “IDN_SPI_SA_SSC_VSR_JN.cpp” shares the overall logical structure used in the program “IDN_SPI_SA_SSC_VSR_RPI.cpp”, except now the ZGB-SDK is used instead of the Boost ASIO Serial Port facility.

Fig. 6.30 gathers the most important Variable and Function Definitions that the reader needs to be familiar with:

- Lines 27 and 28 show that the LN-101 and ZGB-SDK use Port **ttyUSB1** with a Time Out constant equal to 50 ms.
- Lines 199-206 show how Function **send_data()** uses ZGB-SDK’s Functions such as **zgb_tx_data()** to send Remocon packets to the CM-550.
- Lines 47 and 48 indicate that U2D2 is using Port **ttyUSB0** set at 1 Mbps, while Lines 208 and 209 confirm that we will be using SyncWrite/SyncRead packets to work with the Pan-Tilt servos 15 and 16.

```

27  #define DEFAULT_DEVICEINDEX    1 // /dev/ttyUSB1 for ZGB-SDK
28  #define TIMEOUT_TIME          50 // msec

199  void send_data(int message)
200  {
201      if (zgb_tx_data(message) == 0)
202          cout << "Failed to transmit Remocon Packet\n";
203      usleep(5000);
204      stop_once = false;
205      return;
206  }

```

(A)

```

45  #define DXL1_ID                15 // Dynamixel#1 ID: 15 Tilt Servo
46  #define DXL2_ID                16 // Dynamixel#2 ID: 16 Pan Servo
47  #define BAUDRATE               1000000
48  #define DEVICENAME             "/dev/ttyUSB0" // Using U2D2

208  int sync_write_15_16()
209  {

237  int sync_read_15_16()
238  {

```

(B)

Fig. 6.30 Important Variable and Function Definitions in “IDN_SPI_SA_SSC_VSR_JN.cpp”.

The author made a tutorial video summarizing this project available at this web link <https://www.youtube.com/watch?v=7dRmERGYhQY>.

Chapter 7: Enhanced MAX-E2 with RPi4B

Similarly, as for the other robots, the author also tried to mount a Pi Camera on a Pan-Tilt platform to enhance the original MAX-E2 robot. The picture on the left in Fig. 7.1 shows the author's first solution which used two SM-10 servos which turned out to be unsuitable for object tracking as they were "jittery", i.e., unable to hold a set Goal Position reliably.



Fig. 7.1 Two solutions to mount a Machine Vision platform on the MAX-E2 robot.

The picture on the right in Fig. 7.1 shows the author's second and final solution using a 2XL430 servo:

- The Pi Camera was mounted in the Head Piece using a Dummy SM-10, essentially using the ROBOTIS design for the MAX-E1 (Engineer Kit 1).
- This Head Assembly is then attached to the 2XL430 servo using one frame part EF25-F23.

7.1 Hardware/Communication Configurations for RPi4B + E-ME2

Building on the experiences and results gathered from the previous robots, the following hardware configurations for this Enhanced MAX-E2 (E-ME2) robot were implemented (see Fig. 7.2):

- An RPi4B was used as the Co-Controller to the CM-550, as it was obvious that this humanoid robot would not be able to keep its balance when a larger Jetson Nano is strapped to its back.
- The orange "backpack" was designed by ROBOTIS and is available for 3-D Printing at this web link <https://www.tinkercad.com/things/cUMNF5A15eh>. This YouTube video has more details about how to mount this "backpack" along with the RPi4B to the back of this robot (and discussions about other hardware solutions also) - <https://www.youtube.com/watch?v=jjMomyIIIHU>.
- From the author's experience with the E-SPI robot, the LiPo battery option won't work, as one with enough capacity to run 19 servos and the RPi4B will put the E-ME2 out of

mechanical balance. Thus a “tethered-power-cables” option was the only practical solution.

- As a cooling fan was needed, the DXL-HAT could not be used, so the “lighter” U2D2 module was chosen to control the Head Assembly via DXL-SDK.
- A plain USB cable was used to maintain Remocon communications between the RPi4B and the CM-550. That leaves the CM-550’s UART Port available for use with a BT-210 if needed.

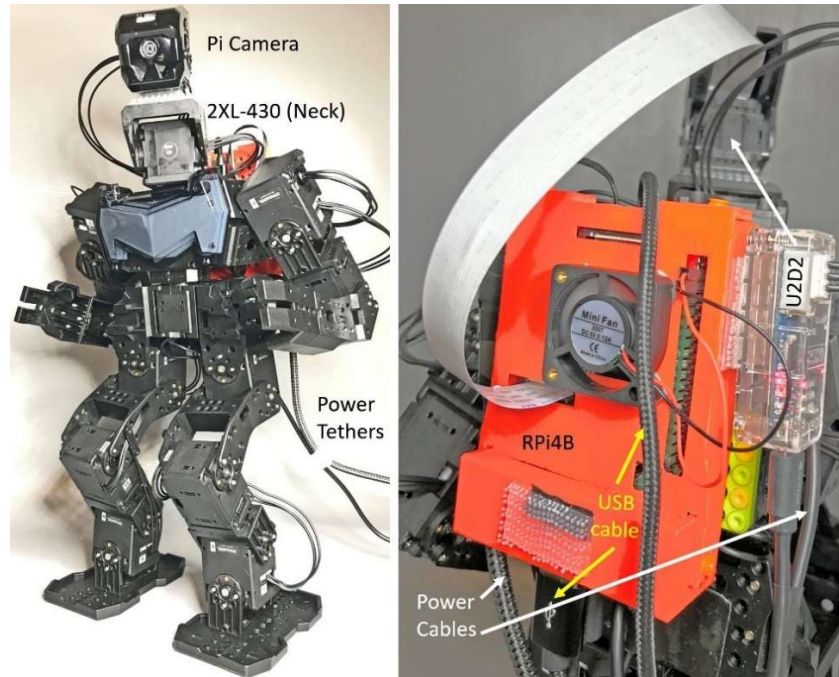


Fig. 7.2 Hardware Configurations used on the E-ME2 robot.

7.2 Motion Offsets Determination

The original MAX-E2 robot comes with an extensive Motions Library, so the obvious goal is to use/reuse these Motions as much as possible with the “new” E-ME2 robot, which has its Center of Gravity shifted to the back due to all the new components attached to its backside and topside (Fig. 7.2). From multiple trials, some successful and some not, the author recommends two essential steps:

1. Reset all servos back to Factory Settings using the MANAGER tool, as they had been modified by the previous projects described in Chapters 1 through 6, also recognizing that the provided Motion Units were created under Factory Reset conditions. The user needs to perform this step with the robot safely lying face down on a level surface to prevent possible damage to the robot.
2. The author found that Motion Unit 1 (Ready Pose) is the best one to use for determining the proper Motion Offsets values to bring the E-ME2 robot “back into balance”, statically and dynamically. The ROBOTIS e-Manual has some basic instructions for this task at https://emanual.robotis.com/docs/en/software/rplustask3/useful_tips/#edit-offset.

7.3 CM-550 Solution in TASK/MicroPython

In this project, 17 servos of the XL430 type are to be controlled by the CM-550 (see Fig. 7.3). Motion Units provided in the ROBOTIS example MAX-E2 MTN3 file will be used/modified to suit the author's needs. The author created both TASK and MicroPython solutions, but only the MicroPython version "IDN_ME2_RC_RPi.py" will be described in this Section.

Fig. 7.6 lists the programming steps used for the familiar Functions `init()` and `init_comm_VSR()`, and the key modifications were:

- Line 13 sets the IMU to its "vertical" orientation.
- Line 14 sets the UART Port to 115.2 Kbps.
- Line 28 sets the CM-550 to expect Remocon Packets through its USB Port.
- Line 30 sets the CM-550 to send Task Print outputs to its UART Port.

```
8 def init():
9     dxlbus.torque_off() # Start modifying EEPROM
10    for i in range(1, 18): # Set DXLs 1 to 17 to Normal Drive & Goal Position mode
11        DXL(i).write8(10, 0)
12        DXL(i).write8(11, 3)
13    DXL(200).write8(15, 0) # Set IMU orientation to "vertical"
14    DXL(200).write8(13, 2) # Set UART Baud Rate to 115.2 Kbps (not used)
15    dxlbus.torque_on() # end modifying EEPROM
16
17    speed = 70
18    motion.speed(speed)
19    motion.play(1) # Basic Pose
20    while(motion.status() == 1):
21        delay(0)
```

```
27 def init_comm_VSR():
28     DXL(200).write8(43, 2)
29     DXL(200).write8(36, 0)
30     DXL(200).write8(35, 1)
```

Fig. 7.6 Functions `init()` and `init_comm_VSR()` used in "IDN_ME2_RC_RPi.py".

7.4 RPi4B Solution in C++

The C++ solution for the E-ME2 project, "IDN_ME2_SSC_VSR_RPi.cpp", is based on the C++ solution for the E-SPI "IDN_SPI_SA_SSC_VSR_RPi.cpp", but the author added a "new" feature for Autonomous Object Tracking but **using the camera and Pan-Tilt platform only**, and **the robot would stand in its Basic Pose** (Motion 1) during this process. This is an example of decoupling functionalities that happen to be designed together previously.

Fig. 7.11 lists the most relevant Variable and Function definitions used in the program "IDN_ME2_SSC_VSR_RPi.cpp":

- The Pan-Tilt servos are now IDed as 19 (Pan) and 20 (Tilt) (Lines 43-44 and 53-54).
- The U2D2 module is used, so Port **ttyUSB0** is enacted at 1 Mbps (Lines 45-46).
- Line 101 defines the "new" Global Boolean Variable **camera_only** and initializes it to FALSE.
- Port **ttyACM0** is now used (wired USB to CM-550) and set to 115.2 Kbps (Lines 116 and 118).

```

43  #define DXL1_ID 19 // Dynamixel#1 ID: 19
44  #define DXL2_ID 20 // Dynamixel#2 ID: 20
45  #define BAUDRATE 1000000
46  #define DEVICENAME "/dev/ttyUSB0" // Using U2D2

53  int IDs[2] = { 19, 20 };
54  int dxl_goal_position[2] = { 2048, 2562 };

99  bool stop_once = false;
100 bool SC_control = true; // "simulated" Supervisory Control for RPi
101 bool camera_only = false; // when true object tracking using camera only

116 const char* PORT1 = "/dev/ttyACM0"; // wired
117 //const char* PORT2 = "/dev/rfcomm0"; // BT c
118 serial_port_base::baud_rate BAUD1(115200);

219 int sync_write_19_20()
220 {

248 int sync_read_19_20()
249 {

```

Fig. 7.11 Selected Variables Definitions in “IDN_ME2_SSC_VSR_RPi.cpp”.

Fig. 7.15 shows the “new” robot behavior when **camera_only** is set to TRUE (Lines 827-831) which is to stand still at the Basic Pose, i.e., “0” Packet behavior (Lines 829-830). Under this condition, the operator would see the robot tracking the color object using the Pan-Tilt platform only. This is a good option to use when the operator is trying out different cameras or enabling the GStreamer option in the RPi4B’s 64-bit OS, when that option is reliably available via the RPi Foundation.

A demonstration video is available at this link <https://youtu.be/bUEz2e15owo>.