

Learning Robotics with
ROBOTIS PLAY Systems
(Excerpts)

By Chi N. Thai

Copyrights 2020 CNT Robotics LLC



CNT Robotics LLC, Lawrenceville

2017

Chapter 1: Introduction

1.1 ROBOTIS System Design Approach

For elementary and middle school students, ROBOTIS carries quite an extensive list of educational robotics kits for its internal market (http://www.robotis.com/index/product.php?cate_code=101010), but the USA and International stores carry a more limited offering (<http://www.robotis.us/robotis-kits/>), essentially the PLAY and DREAM systems. The PLAY system uses the sturdier 7 mm DIA rivet system which is better suited to motor skill levels of younger children. The DREAM system uses a smaller 3.5 mm DIA rivet system which will require more manual skills from the students (see Fig. 1.1).



Fig. 1.1 ROBOTIS' rivet systems: 7 mm (left) and 3.5 mm (right).

The PLAY system (<http://www.robotis.us/play/>) is available in the USA in 3 versions: PLAY300, PLAY600 and PLAY700. The PLAY300 kit uses the older 3.5 mm rivet system and thus is not recommended for the youngest students. The PLAY600 kit is motorized and is quite popular for teaching wheel-based and linkage-based mechanical motions. It has been endorsed by the Global Educator Institute (<http://geiendorsed.com/products/robotis-play-600-pets/>). From Amazon reviews, most 6/7-years old users could handle this kit without adult supervision. Unfortunately, the PLAY600 kit is not programmable. The PLAY700 kit is also motorized with two continuous-turn motors and is equipped with 3 short-range NIR sensors and a speaker (for musical tunes only). The PLAY700 kit is programmable via smartphones and tablets (iOS® or Android®), as well as via Windows® PCs. With the included BT-410 module, it is fully programmable from mobile devices running its R+m.TASK and R+m.PLAY700 Apps and as such can add mobile services such as graphics, audio, video and speech to its robotic repertoire. Additionally, if the user has access to a BT-210 module or a BT-410 USB dongle, then the PLAY700 kit can be used with a Windows PC and interfaced to work with MIT's SCRATCH® 2 software.

The DREAM system (<http://www.robotis.us/dream/>) is a big upgrade over the PLAY700 kit, as it offers a more extensive range of sensors such as Touch, Color, Magnetic, Passive IR and Temperature sensors, and most importantly it can use 2 additional servo motors (not available with the PLAY700 kit) for Position Control applications, for example a pan-tilt connection for pointing a NIR sensor in a wanted direction or a pincer device to grab objects. It also works with the R+m.TASK and R+m.PLAY700 Apps which allow it to additionally access multimedia services on mobile devices.

Although different in their hardware features, both PLAY700 and DREAM systems share a common System Design Paradigm which considers a typical ROBOTIS robot to be a Computer Network with four major components:

- 1) A Hardware Controller, CM-50 for PLAY700 and CM-150 for DREAM (see Fig. 1.2), which is the “Main Brain” for the robot. It contains the user’s instructions for the robot to execute the tasks wanted by the user.

- 2) The Sensors component (e.g. NIR or Touch sensors – see Fig. 1.3), which helps the Hardware Controller get information about its surroundings. The more sophisticated sensors have their own mini-brains to communicate with the Hardware Controller.
- 3) The Actuators component (e.g. Servo Motors or LEDs – see Fig. 1.4), which allows the Hardware Controller to perform the appropriate robot actions upon the real world as programmed by the user. Similarly, advanced actuators such as servo motors have their own mini-controllers to work in concert with the Hardware Controller.
- 4) A Remote Device, which can be an RC-100 module or a smartphone (see Fig. 1.5) or even a Windows PC. The Remote Device has a more flexible role depending on the specific device used and on its current role in this network of robotic components. For example, if the RC-100 is used with the DREAM system, it essentially functions as a wireless Sensor (attached to the user) informing the Hardware Controller about which “Up/Down/Left/Right” buttons had been pressed by the user. When a smartphone (running the R+m.PLAY700 App) is used with the CM-50, it can act as a Sensor telling the CM-50 about which Touch Area on the phone screen had been pressed by the user, or it can act as an Actuator displaying appropriate graphics or texts as commanded by the CM-50 according to its programmed logic. When a PC is used as a Remote Device, the issues become more complex. For example, if TASKS Virtual Controller (see Fig. 1.6) is used, then the PC acts as a glorified Sensor informing the Hardware Controller CM-50/150 about which keyboard buttons had been pushed by the user. However, if the SCRATCH 2/R+SCRATCH tool chain is instead used on the PC, then the PC takes on the role of a “true” Hardware Controller, as SCRATCH 2 codes run on the PC and not on the CM-50 controller which now has a diminished role, i.e. just letting SCRATCH 2 commands from the PC to pass through to Actuators and Sensors via the CM-50’s Firmware (see Fig. 1.8).



Fig. 1.5 Selected Remote Devices: RC-100 (Left) and Smartphone (right).

To achieve hierarchical control in this adaptive network of robotic components, **ROBOTIS software uses two types of communication packets: Instruction and Status.** Instruction packets can only be issued by the Top-Level Controller which can be either the Hardware Controller (CM-50/150) or the Remote Device, and there can be only ONE Top-Level Controller at any one time (if ones use the proprietary firmware provided by ROBOTIS for its controllers). Once a network component receives an Instruction packet, it will execute the prescribed instruction and will optionally send back Status packets to the originator of that Instruction packet.

Fig. 1.7 shows a configuration of network components and communication packets flow for the situation where the Top-Level Controller is the Hardware Controller (CM-50/150) using the RoboPlus software suites

(i.e. MANAGER, TASK, MOTION), while the Remote Device could be any of the following devices: physical RC-100, virtual RC-100 on the PC, smartphone or tablet using the R+m.PLAY700 App. In this configuration, please note that only the Hardware Controller can issue Instruction packets and that Actuators and Sensors can never issue Instruction packets as they are at the “bottom” of the control hierarchy.

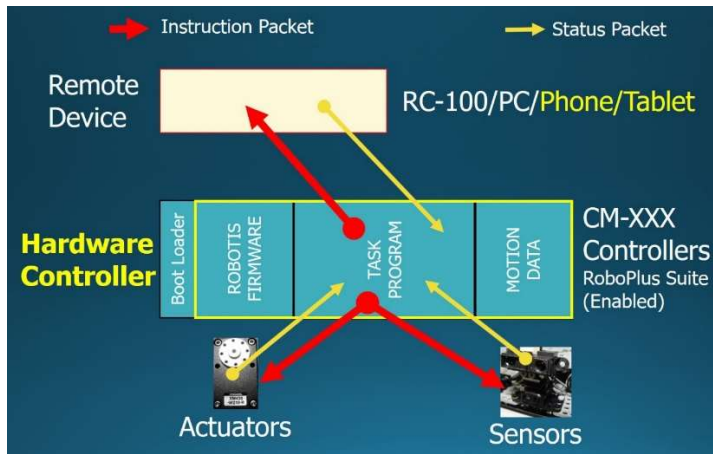


Fig. 1.7 Hardware Controller as Top-Level Controller.

Fig. 1.8 shows a second network configuration where the Top-Level Controller is the Remote Device which can be a PC using the SCRATCH 2/R+SCRATCH tool chain or a mobile device using a more advanced ROBOTIS software tool called OLLOBOT SDK (which is beyond the scope of this book and interested readers are referred to Thai (2017) for more details). In this configuration, please note that all TASK and MOTION functionalities would now be disabled on the CM-50/150 (mainly because ROBOTIS controllers’ firmware are written this way). Please also note that all communication packets are now routed through the R+SCRATCH application on the PC and the “Firmware” component of the controller before reaching the Actuators and Sensors, meaning that communication delays would likely occur for this configuration due to the extra layer of communication software (especially when serial communication through R+SCRATCH is restricted to a rate of 57.6 Kbps).

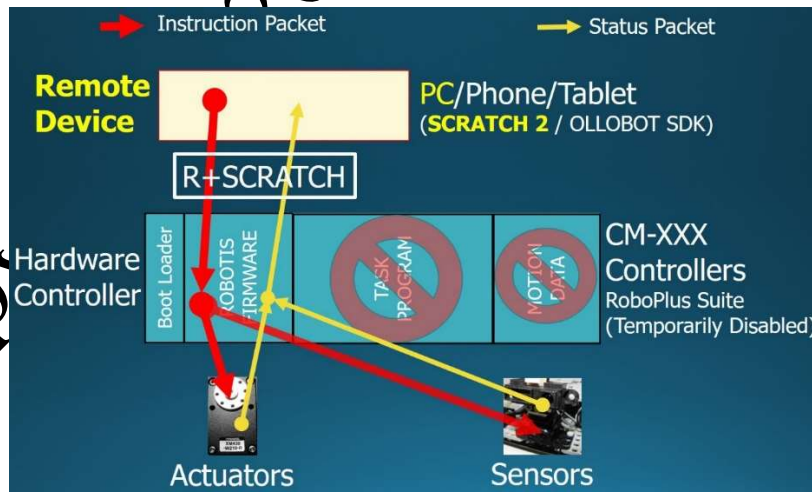


Fig. 1.8 Remote Device as Top-Level Controller.

1.2 Sense-Think-Act Paradigm

Usually the younger robotics students were surprised when I started my robotics short courses by mentioning that they had been using the Sense-Think-Act paradigm in everyday activities by themselves. I used Fig. 1.9 to explain how humans use our Senses to let us know about the external World (i.e. Perception) and apply those Sensations into our Cognitive process (i.e. Thinking) to devise proper Actions/Reactions to the situation at hand. These Actions/Reactions in turn would change some aspects of the external World resulting in new Sensations triggering the next World-Human-Interactions cycle and so forth.

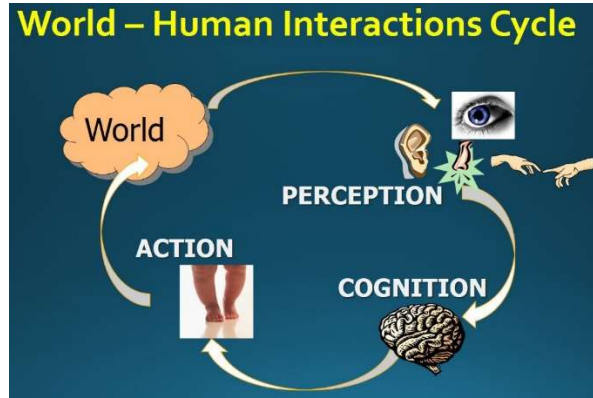


Fig. 1.9 World-Human Interactions Cycle.

Thus, in a way, doing Robotics is just trying to reproduce this interactions cycle onto a robot as shown in Fig. 1.10. Mechanical or electronic Sensors would convert their interactions with the World into Input Data to be sent to the Robot/Computer which was previously Programmed by the user to deal with Events deemed important to the operation of the Robot. Using this programmed Logic, the Robot Outputs appropriate commands to its Actuators which then could change the Sensors' perspective onto the World and therefore generate new Input Data for the Robot/Computer to process anew, through another interactions cycle.

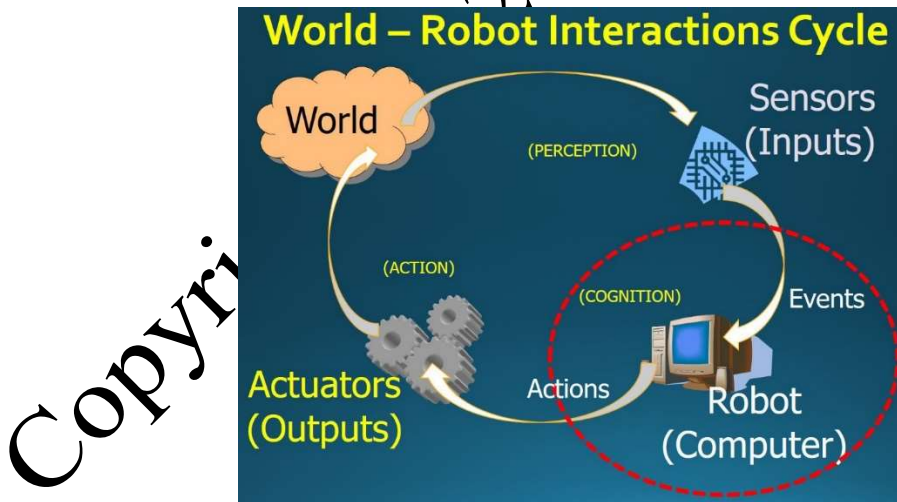


Fig. 1.10 World-Robot Interactions Cycle.

Let's go through an example to see how this Sense-Think-Act paradigm can help us solve the following challenge as described in Fig. 1.11, where a CM-50 based CarBot would be able to follow an irregular black track in the shape of the number 8. Furthermore, if the CarBot encounters a frontal obstacle during its travel,

it would need to avoid it by swinging around until it gets back onto the track but in the other direction. Next, the CarBot should continue its auto-tracking mission unless it encounters a frontal obstacle again.

Fig. 1.11 shows that the CM-50 has three IR Sensors whereas the Center IR could be used to detect a frontal obstacle and the Right and Left IR sensors could be used to detect the curvy black track. The CM-50 also has a Motor on each Left and Right side, so these motors should be able to be programmed to provide all CarBot maneuvers that would be needed to perform the challenge successfully (i.e. Forward, Turn Left and Turn Right). This System Design solution seems feasible enough, but to progress further towards the actual coding of the CM-50, we need to zoom in around the Robot/Controller area in Fig. 1.10 (i.e. the dashed red boundary in the lower right corner). The Sense-Think-Act paradigm still applies at this level, but some robotics practitioners would start using different labels such as “Events”, “Actions” and “Reactive Control” as they are more useful in the next steps that we need to take towards the creation of the actual computer code to control this CarBot. Interested readers should read Matarić (2007) for a more complete description of other robotics control approaches.

With the “Reactive Control” approach, ones need to match up a possible Condition/Event with an appropriate Action for the robot to do (see Fig. 1.12).

| Reactive Control Approach | |
|---|--|
| Given Condition (Event) >> Appropriate Robot Action | |
| Condition/Event | Action |
| Car on top of Track | Car Goes Forward |
| Car left of Track | Car Turns Right |
| Car right of Track | Car Turns Left |
| Frontal obstacle detected | Car Swings Right until on top of Track again |

1 and Only 1 Condition (Event) can happens at any 1 time.
Multiple Conditions (Events) can happen at any 1 time.

Fig. 1.12 Reactive Control Approach used for CarBot challenge.

For example, if the CarBot happens to be lined up on top of the track already, then it should just go forward. There are 3 other Event/Action pairs listed in Fig. 1.12, and are these four pairs the only ones to consider or needed for this robot challenge? My advice for beginning roboteers is just to start with a reasonable set of Event/Action pairs and then improve on them as needed in later refinement steps.

But how would a beginner in robotics comes up with such a list as shown in Fig. 1.12 in the first place? Here, in keeping with the introductory scope of this book, the author offers a simple “thought experiment” or “visualization exercise” approach:

- 1) First we would imagine to be “one” with the robot, so our Left and Right Eyes would match with the Left and Right IR Sensors, i.e. we keep our nose to the ground so that we can see the black track (to be exact, only a partial view of it). However, humans do not have a third eye right on top of our head to correspond to the Center IR, so we would really have to use our imagination and use our “mind’s eye” as the third eye there!

Chapter 2: Getting Started

The PLAY700 kit, as received, is fully functional with a mobile device on iOS® or Android® OS. This web site (<http://www.robotis.us/software/play700/>) contains download links for all needed PLAY700 software and tutorial videos. One can use the mobile R+m.TASK and R+m.PLAY700 Apps using Bluetooth 4 on the mobile device and a corresponding BT-410 module on the PLAY700 robot. Some features of the R+m.PLAY700 tool are only functional on an Android device, and not on an iOS device (see Section 2.1).

If the user wants to use MIT's SCRATCH 2 software with PLAY700, then the user is restricted to Windows PCs, as the needed ROBOTIS helper software named R+SCRATCH is available for Windows OS only. The user also needs to install the SCRATCH 2 Offline Editor (<https://scratch.mit.edu/download>) on the PC. The user is further required to purchase a USB BT-410 dongle from ROBOTIS (or their vendors) so that the Windows PC can communicate with the BT-410 on the robot (see Fig. 2.1).

Another handy ROBOTIS software tool called MANAGER also runs on Windows PCs only. It provides a quick way to check if the CM-50 is working OK hardware wise, and to update its firmware when necessary.

In this Chapter, we'll use a BasicBot as the demonstration platform (see Fig. 2.1). Please watch [Video 2.1](#) to see how it is put together.



Fig. 2.1 BasicBot demonstration platform.

2.1 Using PLAY700 kit on Mobile Devices

The first step is for the reader to install the R+m.TASK and R+m.PLAY700 Apps on his or her chosen mobile device. These Apps are available at the appropriate App Stores (<https://itunes.apple.com/us/developer/robotis-co-ltd/id948481761>) or (<https://play.google.com/store/apps/developer?id=ROBOTIS&hl=en>).

Next, the reader is invited to view [Video 2.2](#) which is a tutorial on how to:

- 1) Connect the mobile device to the BT-410 on a BasicBot platform. It also demonstrates other setup settings (see Fig. 2.2).
- 2) Explore the features available in a typical PLAY700 project and see the differences in capabilities between Android and iOS devices (see Fig. 2.3).
- 3) Use the TASK tool to have a quick browse at the preloaded PLAY700 demo program, and show how to download it to the CM-50 and run it in concert with one of the 4 R+m.PLAY700 example projects on the mobile device (see Fig. 2.4).

2.2 Using PLAY700 kit on Windows PC

Assuming the reader had installed successfully the following software packages on the PC:

- 1) MANAGER (<http://www.robotis.us/roboplus2/>).
- 2) TASK (<http://www.robotis.us/roboplus2/>).
- 3) R+SCRATCH (<http://www.robotis.us/roboplus2/>).
- 4) MIT SCRATCH 2 (<https://scratch.mit.edu/download>).

The MANAGER tool is used when the user needs to check if a given CM Controller and the actuators or sensors attached to it are working properly. Fig. 2.5 shows the main menu for the MANAGER tool when it connects to a CM-50 (i.e. the controller for the PLAY700 system). The column labeled “Control Table” contains various functions that can be tested directly via MANAGER: for example, AUX LEDs, Ports 1 & 2 Motor Speeds, etc... These functions can also be programmed via the TASK tool as shown later. Sometimes, the CM-50 may malfunction in some way, and usually the best solution is to “recover its firmware” which can only be done via this MANAGER tool. Please see [Video 2.3](#) for more details.

The TASK tool is THE “programming” tool, where the results from Input/Output tables, created from steps described in Chapter 1, are converted into TASK program statements. The TASK tool may be described as a context-sensitive line editor (see Fig. 2.6).

It has special “Smart Device” functions that interface with the R+m PLAY700 App allowing the CM-50 to control various services on the mobile device such as graphics, audio, video, text-to-speech, speed recognition and image processing (see Fig. 2.7). Please see [Video 2.4](#) for a quick tour of its capabilities. Its proficient use requires lots of training and practice on the part of the user which will be developed in Chapter 3.

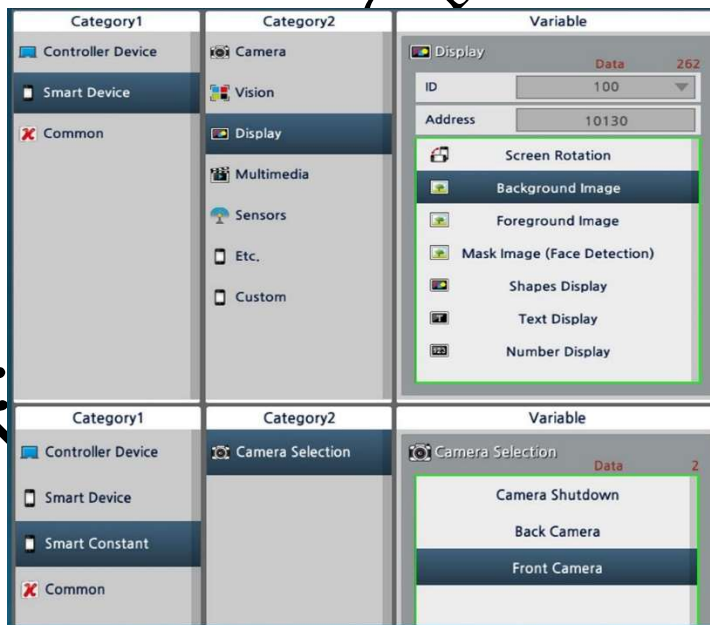


Fig. 2.7 SMART DEVICE functions available inside TASK tool.

R+SCRATCH (see Fig. 2.8) is an important background application that interfaces with the SCRATCH 2 software tool, to allow SCRATCH 2 commands to be sent from a PC communication port to reach into the actuators and sensors on the CM-50. Please watch [Video 2.5](#) for a demonstration of its typical use.

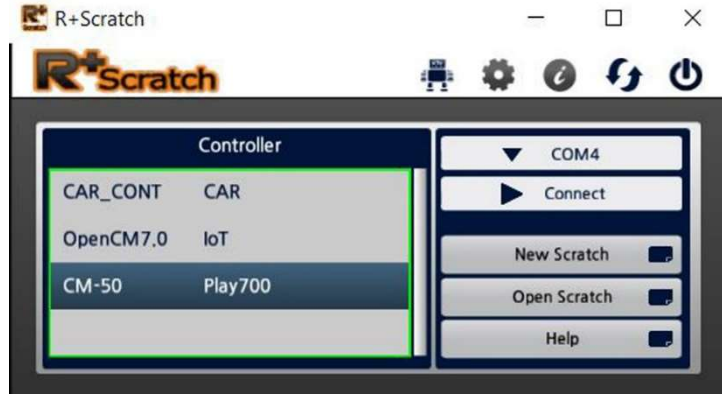


Fig. 2.8 R+SCRATCH background application for PC.

SCRATCH 2 is a well-known software tool with users world-wide (https://wiki.scratch.mit.edu/wiki/Scratch_Statistics#Scratchers_Worldwide). The SCRATCH 2 tool may be described as a block-programming tool (see Fig. 2.9). Please see [Video 2.8](#) for a quick tour of its event-programming capabilities which are most applicable to robotics. Its proficient use requires lots of training and practice on the part of the user which will be developed in Chapter

TT ROBOTICS LLC

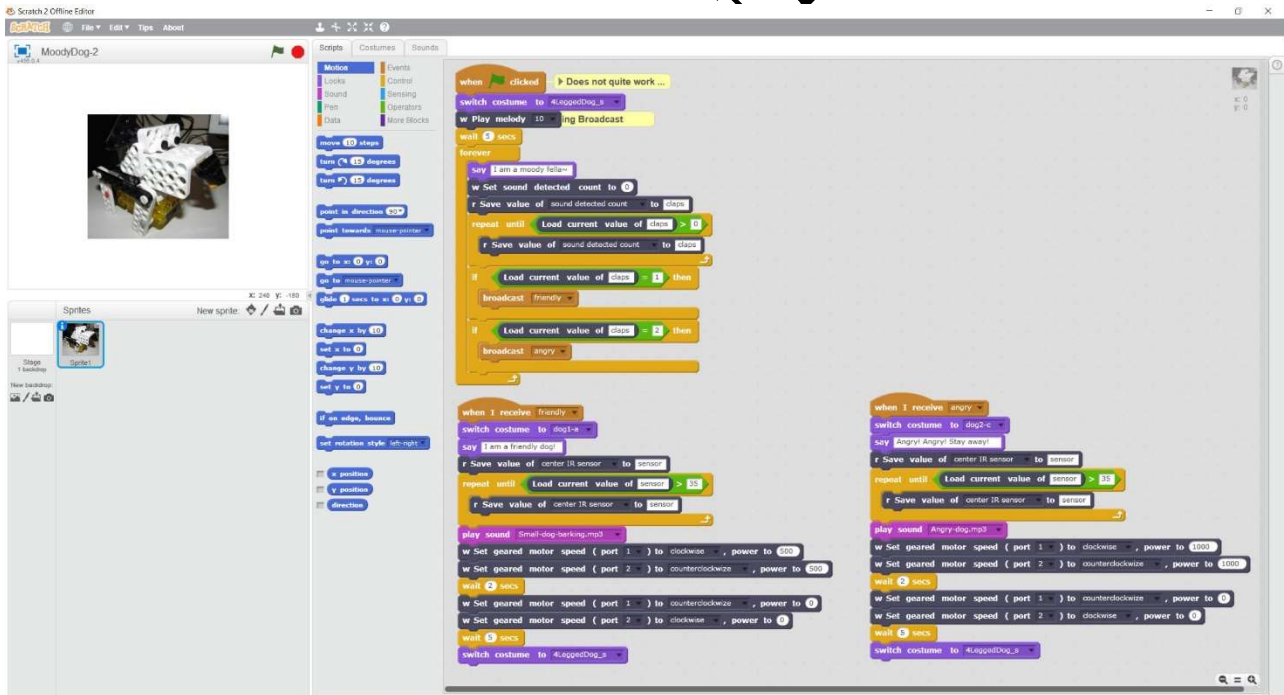


Fig. 2.9 SCRATCH 2 for Moody DogBot project.

2.3 Building “Spinning Top” robot

Let’s build “Spinning Top” which will serve as the standard platform from where we can develop foundational programming skills using the TASK, R+m.PLAY700 and SCRATCH 2 tools (see Figs. 2.10 and 2.11, and watch [Video 2.7](#) to see how “Spinning Top” was constructed from “BasicBot”). We’ll use this robot to learn about robotics interfacing and control techniques in Chapters 3 and 4.

2.4 Downloading Example Codes

For PCs and Android devices, the reader needs to first visit this DropBox link (https://www.dropbox.com/s/mecq2yuvsrfixfr/LearningRoboticsWithRobotisPlaySystems_Codes.zip?dl=0) and download the ZIP file to the Windows PC, next unzip it and transfer all the example codes from the PC to the Android device via Windows Explorer (please see Part 1 of [Video 2.8](#)). For iOS devices, the procedure is more involved as the Windows PC normally can only “see” the Photo Gallery on these devices. The author uses the “iMazing” software which can transfer seamlessly the example codes from the PC over to the iPhone file directory (see Part 2 of [Video 2.8](#)). The author has not done a thorough Web search to see if there are some “free” software/app that can do similar files transfer tasks as with iMazing.

Copyrights 2020 CNT Robotics LLC

Chapter 3: Using R+TASK & R+m.PLAY700

This chapter is written for a complete beginner in robotics and computer programming, thus if the reader already had some experiences in those areas, please bear with the “pedestrian” approach used and feel free to skip to the more advanced sections using the R+m.PLAY700 mobile App as needed. We will be using the “Spinning Top” platform (Fig. 2.10 from Section 2.3) to learn how to use the TASK tool and the R+m.PLAY700 App. A “Spiral” learning approach (Bergin, 2012) will be used for instructional purposes where the author would first show how to use the TASK tool with the “plain” CM-50 to perform a certain project. Next, the author would “enhance” it with similar or enhanced features provided by the R+m.PLAY700 App (whenever appropriate).

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

- 1) ASSIGNMENTS – TASK provides two types of assignment statements: LOAD and COMPUTE.

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

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

- 2) LABEL and JUMP statements are supported.

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

- 4) LOOPS – for Repetition statements, TASK provides “WAIT WHILE”, “LOOP WHILE”, “LOOP FOR”, “ENDLESS LOOP” and “BREAK LOOP”.

- 5) FUNCTION definition and calling are provided in TASK. A special CALLBACK function can also be used: it is triggered by a hardware timer every 8 ms independently of the main TASK program.

3.1 Learning Sequence Control in TASK programs

Please review [Video 2.4](#) if you had not done so to refresh your memory about navigating the menus of the TASK tool on a PC or mobile devices.

In this unit, we are going to learn that by default the CM-50 controller behaves sequentially, i.e. it can do only one thing at a time. [Video 3.1](#) shows how to create a TASK program from scratch, please save this program as “SequenceControl-1.tskx”. [Video 3.1](#) also shows how this work was done on PCs and on mobile devices (Android and iOS). Alternately, the reader can use the TASK tool to open the same pre-written program from the provided ZIP file containing all the example codes for this book (see Section 2.4). Fig. 3.1 is a screen capture of this program which has only 3 lines of “relevant” code (lines 5-7). More formally, they are called “Statements”. All TASK program begins with the “START PROGRAM” statement

(line 3) and an opening curly bracket “{” statement (line 4), and ends with a closing curly bracket “}” statement (line 8).

Another way to look at the code in Fig. 3.3 is to treat a “WAIT WHILE” as a means for the Controller to be “inactive” during certain time periods (lines 9, 12 and 15), however during those same time periods the Robot was quite “active” as its actuators would still be executing the last commands given to them by the Controller (lines 11 and 14). **In other words, a “robot” does not have to be physically “inactive” whenever the “computer/controller” is set to be logically “inactive”.** This “controlling/programming” scheme is used quite often in robotics and will be shown again in later examples in this book.

3.2 Learning Program Modularization with Functions in TASK programs

Let’s start with the “Maneuvers-1.tskx” program (Fig. 3.4) which uses previous concepts already explained in Section 3.1. But this time, they are combined to make the robot go forward at Speed=512 for 0.5 s. and a COMPUTE type of statement is used to set separately the DIRECTION and SPEED for each of the Motor Ports 1 and 2 (lines 8 and 9 in Fig. 3.4).

```

5      Speed = 512
6      DelayTime = 0.500sec
7
8      PORT[1]:Geared Motor = CW:0 (0.00%) + Speed
9      PORT[2]:Geared Motor = CCW:0 (0.00%) + Speed
10     High-resolution Timer = DelayTime
11     WAIT WHILE ( High-resolution Timer > 0.000sec )

```

Fig. 3.4 Program “Maneuvers-1.tskx”.

The COMPUTE statement is of the form “A = B Op C” which has the now familiar Assignment operator “=” separating the RHS expression (i.e. “B Op C”) and LHS (i.e. “A”). The RHS now has two Operands “B” and “C” linked by a second operator Op which can be of 7 types:

- “+” standing for the arithmetic addition operation (used in lines 8 and 9).
- “-” standing for the arithmetic subtraction operation.
- “*” standing for the arithmetic multiplication operation.
- “/” standing for the arithmetic division operation.
- “%” standing for the remainder operation (i.e. yields the “remainder” when doing “manual” integer division).
- “&” standing for a bit-wise logical AND operation.
- “|” standing for a bit-wise logical OR operation.

A “COMPUTE” statement is used here (line 8 of Fig. 3.4) instead of a “LOAD” statement (like in line 11 of Fig. 3.3) because SPEED is now a Variable Parameter, currently set to 512 but it can be changed to something else in other parts of the computer program (to be shown later in this Section 3.2). If the need was for a fixed numerical constant that would not change throughout the robot program, then a “LOAD” statement would be more efficient to use. [Video 3.6](#) shows how to edit COMPUTE statements inside the program

“Maneuvers-1.tskx”, how this program perform at run-time and how to modify it into “Maneuvers-2.tskx” which uses a new type of computer programming structure called “FUNCTION” (see Fig. 3.5).

The later part of [Video 3.6](#) showed how to use the MAIN PROGRAM area as a “sandbox” area where the user can try out different coding ideas, while the FUNCTION areas were used as a repository of coding ideas that had worked well. This is a “shallow” understanding/application of the FUNCTION structure which serves as a useful but intermediary step towards a more complete understanding of the issues later.

Once the basic Functions “Forward”, “Backward”, “Right” and “Left” are defined, the program “Maneuvers-3.tskx” shows how to use them to create a sequence of robot maneuvers, perhaps needed to negotiate a maze, e.g. going forward, turn left then going forward again (see Fig. 3.6). These tasks were done by CALLing those predefined Functions “Forward” and “Left” as needed (lines 9, 13 and 17).

Part 1 of [Video 3.7](#) explains the details of how the controller jumped from a CALL statement in the Main program (such as Line 9 in Fig. 3.6) into the Function definition area, executed the statements defined in this Function, then returned to the Main program to execute the next statement (e.g. Line 10).

The image shows a screenshot of a program editor with a line number column on the left (4 to 20) and a code area on the right. The code is as follows:

```
4 START PROGRAM
5 {
6   Speed = 512
7   DelayTime = 0.500sec
8
9   CALL Forward
10  High-resolution Timer = DelayTime
11  WAIT WHILE ( High-resolution Timer > 0.000sec )
12
13  CALL Left
14  High-resolution Timer = DelayTime
15  WAIT WHILE ( High-resolution Timer > 0.000sec )
16
17  CALL Forward
18  High-resolution Timer = DelayTime
19  WAIT WHILE ( High-resolution Timer > 0.000sec )
20 }
```

Fig. 3.6 Program “Maneuvers-3.tskx”.

The reader can see that by using Functions the logic flow in the Main program is easier to understand as it is presented in a more modular fashion, perhaps “logically coarser”, while pushing the “gory” operational details into the Function definitions themselves. For example, this CM-50 robot has only two motors, but let’s say that you move up to the DREAM system so your robot now has four motors to control instead of two, and you still need to perform the same “mission” (forward/left/forward). In this case, you would not have to modify the main program at all, but you will need to add new statements to take care of Motors 3 and 4 inside the appropriate Functions “Forward”, “Backward”, “Left” and “Right” (see Part 2 of [Video 3.7](#)).

Another useful application of Functions is to reduce the number of groups of repeating statements in the Main program. Fig. 3.6 shows three groups of the “same” statements involving the Hi-Res Timer (e.g. lines 10-11, 14-15 and 18-19). The program “Maneuvers-4.tskx” shows how to define the Function DELAY (Fig. 3.7) just one time and how to CALL it in the Main program as many times as needed (Fig. 3.8). “Maneuvers-4.tskx” also shows how to use the Variable Parameters Speed and DelayTime to create maneuvers with varying speeds and delay times, i.e. a more adaptive algorithm to deal with complicated mazes.

In a given function, just before calling that function: for example, set up Speed (line 8) before calling Function Forward (line 9). [Video 3.8](#) shows the runtime performance of the program “Maneuvers-4.tskx”.

In this Section 3.2, the robot’s maneuvers were “timed maneuvers”, thus they may not work consistently for different/changing environments such as different ground surfaces that the robot may run on or the charge level of its batteries (low or high). In later sections, we’ll use more “conditional” programming techniques so that the same robot code can work in different environments than the one that the code was originally created in.

3.3 “Spinning Top” Maneuvers using IR Center Sensor

We are now ready for our first implementation of the Sense-Think-Act paradigm previously discussed in Chapter 1. The sensing component will be the IR Center Sensor of the “Spinning Top” robot as shown in Fig. 3.9.



Fig. 3.9 “Spinning Top” robot and its IR Center Sensor.

The tasks assigned to this robot are as follows:

- 1) At start, the robot plays a musical melody to let the user know that it is “ready for action”.
- 2) Next, the behaviors of the robot depend on the position of an object relative to the IR Center Sensor, thus the condition-action list to be considered is as follows:

If the IR Center Sensor reading is less than 50, the robot stays still.

If the IR Center Sensor reading is between 50 and 300, the robot spins right with its speed proportional to the IR Center Sensor reading and it plays a musical scale for 0.1 second.

If the IR Center Sensor reading is between 300 and 500, the robot spins left with its speed proportional to the IR Center Sensor reading and it plays a different second musical scale for 0.1 second.

If the IR Center Sensor reading is larger than 500, the robot rolls forward at Speed=512 to escape from the object and it plays a third musical scale for 0.5 second.

This condition-action list indicates that we are dealing with a situation whereas one and only one condition can happen at any one time during the runtime performance of this robot, thus we will use the IF-ELSE-IF structure for our solution to this problem. This challenge will be solved in two ways:

- 1) Using the built-in musical resources of the CM-50.

2) Using the text-display and musical functions available on a smartphone via the R+m.PLAY700 App.

3.3.1 Solution using CM-50's built-in speaker

The program “IR-based-Maneuvers.tsk” describes the solution approach used:

Please note that according to the considered condition-action listing, the controller is supposed to check if IR_Sensor is between 50 and 300 at line 18, but it uses a condition that involves only checking for (IR_Sensor <= 300). Is that sufficient to do what is required? And the answer is that it is OK to just check for the condition (IR_Sensor <= 300) here, because the condition (IR_Sensor > 50) was implicitly TRUE when line 18 was chosen (see previous paragraph). Thus if (IR_Sensor <= 300) turns out to be TRUE (meaning that IR_Sensor is between 50 and 300), the controller assigns the value of IR_Sensor to Speed and calls the Function Right to make the robot spin right at a Speed proportional to IR_Sensor's value. The controller also plays Do# for 0.1 second, then exits this IF-ELSE-IF structure and makes another loop starting at line 13. However, if the condition (IR_Sensor <= 300) turns out to be FALSE, i.e. (IR_Sensor > 300), then the controller goes on to execute the next ELSE-IF statement at line 26.

Similarly, executing line 26 means that the controller is checking to see if IR_Sensor is between 300 and 500. If this is TRUE, then the robot spins left with a Speed proportional to IR_Sensor's value, plays Re# for 0.1 second, exits this IF-ELSE-IF structure and continues with its looping task starting at line 13. If the condition in line 26 turns out to be FALSE (i.e. (IR_Sensor > 500)), then line 34 is executed next.

3.3.2 Solution using Smartphone services: Text-Display, Text-to-Speech and Music-Play

In this section, the program “IR-based-Maneuvers.tskx” is adapted to make use of the Text-Display, Text-to-Speech and Instrument-Play services available on smartphones and tablets which are herein used as advanced remote devices. On TASK programs, these multimedia services are accessible via the SMART DEVICE and SMART CONSTANT categories and the resulting program is called “IR-based-Maneuvers-App.tskx”. The user also needs to prepare a matching PLAY700 project on the Remote Device using the R+m.PLAY700 App (see Part 1 of [Video 3.10](#)), whereas 5 Text Items are set up (See Fig. 3.12).

| Text | Add |
|-----------|-----|
| 1 Ready! | |
| 2 Stopped | |
| 3 Right | |
| 4 Left | |
| 5 Forward | |

Fig. 3.12 Five Text Items used by program “IR-based-Maneuvers-App.tskx”.

The program “IR-based-Maneuvers-App.tskx” starts out in the same way as shown in Fig. 3.10 (i.e. it plays Melody 2 out of the CM-50’s speaker). But the next actions differ because we are now dealing with the issue of waiting on the Bluetooth connections to be made securely between the Remote Device (i.e. R+m.PLAY700 App) and the Robot (i.e. TASK program). Fig. 3.14 shows the next group of statements:

Lines 10 and 11 set up the controller to listen for any Sound Clap before quitting the WAIT WHILE statement in line 11.

Thus, once the user creates some loud noise, the controller executes line 12 to set the Remote Device’s Display into landscape mode.

Line 13 clears the Display, and line 14 displays Text Item 1 (i.e. “Ready!”) at position (3,3) (i.e. at the Display’s center), using a Text Size of 100 and White for Text Color.

Line 15 uses the Text-to-Speech engine to voice the Text Item 1 “Ready!”. Line 16 uses a WAIT WHILE loop to wait for the speech engine to finish its job before continuing with the rest of the code. Please note that the Text-to-Speech engine cannot make the difference between “Ready” and “Ready!” at all (i.e. no intonation implemented).

```

9 // User needs to make sure that robot is connected to mobile device. User claps hands when ready.
10 Result of Sound Counter = 0
11 WAIT WHILE ( Result of Sound Counter == 0 )
12 SMART: Screen Rotation = Landscape Mode (2)
13 SMART: Text Display = 0
14 SMART: Text Display = [Position:(3,3),[Item:1],[Size:100],[Color:White]
15 SMART: Text to Speech (TTS) = Text Item 1
16 WAIT WHILE ( SMART: Text to Speech (TTS) != 0 )

```

Fig. 3.14 Setting up Smart Device to display and voice out Text Item 1,

Fig. 3.15 shows the beginning of the Main Endless Loop in charge of monitoring the IR Center Sensor and deciding on the appropriate behavior(s) for the robot to perform, depending on the real-time value of that sensor:

Similarly, as before, the controller reads the IR Center Sensor and saves the value found there into Parameter “IR_Sensor” (line 20).

Then it enters the IF-ELSE-IF structure at line 21 to figure out if the object is present but still a way from the NIR Sensor itself (i.e. IR_Sensor <= 50). If this is TRUE, it calls Function Stop (line 23) and displays Text Item 2 (e.g. “Stopped”) on the Display Screen’s center in Red with a letter size of 100 (line 24). If this condition turns out to be FALSE (i.e. IR_Sensor > 50), then it would transfer to line 26 to check on the next ELSE-IF statement (see Fig. 3.16).

3.4 “Avoider” Maneuvers using Left and Right IR Sensors

In this section, we’ll use the same robot platform as shown in Fig. 3.9, but it is renamed “Avoider” to emphasize the types of maneuvers described herein. The “Avoider” will be using its Left and Right IR Sensors to detect a single object and to accomplish tasks described in the following condition-action list:

If there is no obstacle in front of Avoider, it can go forward.

If there is an object on the Left of Avoider, it should back away and turn to the Right to avoid the obstacle.

If there is an object on the Right of Avoider, it should back away and turn to the Left to avoid the obstacle.

If there is an object in front of Avoider, it should go straight back to avoid the obstacle.

At first analysis, this seems to be a familiar programming situation of choosing one and only one alternative among several possible alternatives, thus an IF-ELSE-IF structure should do an adequate job when combined with an Outer Endless Loop. The TASK solution is named “Avoider-1.tskx”. It starts with the playing of Melody 2 (same code as shown in Fig. 3.10). The main algorithm is described in Fig. 3.18:

The Endless Loop starts out by reading in the latest values from the IR Left and Right Sensors and saves them respectively into Variable Parameters IR_Left (line 14) and IR_Right (line 15).

Next is an IF-ELSE-IF structure with 4 mutually exclusive cases:

At line 16, if the values of Parameters “IR_Left” and “IR_Right” are both less than 10 (i.e. no obstacle is in front of the robot), the robot can go forward and the controller would then exit the IF-ELSE-IF structure and restart another loop at line 14. If this condition is FALSE, i.e. when either “IR_Left” or “IR_Right” have values larger than 10, or in other words when either IR sensor sees an object, the controller would need to check further by executing the next ELSE-IF statement at line 20.

The condition $((IR_Left \geq 100) \text{ AND } (IR_Right < 10))$, used for the ELSE-IF statement at line 20, represents a situation where there is an object on the left of the robot and none of its right side. If this condition evaluates to TRUE, the robot would turn right by backing away to avoid the obstacle on its left (line 22). The controller next exits this IF-ELSE-IF structure, and loops again at line 14. If this condition evaluates to FALSE, then line 24 is executed next.

Similarly, the condition $((IR_Right \geq 100) \text{ AND } (IR_Left < 10))$, used for the ELSE-IF statement at line 24, represents a situation where there is an object on the right of the robot and none of its left. If this condition evaluates to TRUE, the robot would back away and turn left to avoid the obstacle on its right (line 26). The controller next exits this IF-ELSE-IF structure, and loops again at line 14. If this condition evaluates to FALSE, then line 28 is executed next.

The ELSE-IF statement at line 28 uses this condition $((IR_Left \geq 100) \text{ AND } (IR_Right \geq 100))$ to represent a situation when there is an object close enough to the robot’s front such that both IR Sensors report data values larger than 100. If this condition is TRUE, the robot would roll straight back away from the obstacle, otherwise the controller loops back to line 14.

The next program “Avoider-3.tskx” offers a different solution technique to these maneuvering tasks. **This new approach still relies on the same condition/action listing as described at the beginning of this section 3.4, but it distills different behavior patterns from this list:**

- 1) We must identify a “default” action which in this case is “Going Forward”.
- 2) We will be using several simple IFs in parallel instead of a single big IF-ELSE-IF structure like before. **We also need to do a deeper analysis of the condition/action list to retain only the “essential” condition/action pairs.**
- 3) We also **need to apply a Delay Time to each action made** (this part was not needed when the IF-ELSE-IF structure was used). This Delay Time can be tuned for optimal performance.

Fig. 3.20 shows the relevant section of the algorithm used:

Line 10 showed that a Delay Time of 100 ms was used (the reader should try different values such as 0, 50, 150 etc... to see the effect on the overall performance of the robot).

An ENDLESS LOOP is still used (line 13) and starts by updating the parameters IR_Left and IR_Right with the latest data input from the IR Left and Right Sensors (lines 15 and 16).

The first IF statement (lines 17-21) represents the condition when there is an object on the robot's left (line 17), then the robot would back up and turn right (as before) to steer away from the obstacle (line 19). A short delay (line 20) is necessary for the robot to stay in this "action" to make this algorithm work properly.

The second IF statement (lines 22-26) represents the condition when there is an object on the robot's right (line 22), then the robot would back up and turn left (as before) to steer away from the obstacle (line 24). A short delay (line 25) is also necessary for the robot to stay in this "action" to make this algorithm work properly.

Please note that the "default" maneuver is always programmed as the "Last Action" "Going Forward" in this case – line 27).

```
9      Speed = 256
10     DelayTime = 100
11
12     // Main Loop
13     ENDLESS LOOP
14     {
15         IR_Left = IR Left
16         IR_Right = IR Right
17         IF ( IR_Left >= 100 )
18         {
19             CALL RightBack
20             CALL Delay
21         }
22         IF ( IR_Right >= 100 )
23         {
24             CALL LeftBack
25             CALL Delay
26         }
27         CALL Forward
28     }
```

Fig. 3.20 Alternate Maneuvering Algorithm using "Parallel IFs" in program "Avoider-3.tsx".

It is important to perform a "thought experiment" here to "preview" the possible results of this Endless Loop at run time:

Every time that the Endless Loop is executed, line 27 will be executed, i.e. the robot will always go forward (a little bit), regardless of everything else that may happen.

Regarding the two parallel IFs, during some execution cycle(s), none of them could be activated (i.e. when there is no obstacle), or one of them could be triggered (i.e. when the obstacle is on either side of robot), or both could be triggered (i.e. a frontal obstacle). The alert reader may have noticed that seemingly the case for "a frontal obstacle" was not dealt with in this algorithm, so how would the robot behave at run time when encountering a frontal obstacle? Watching [Video 3.13](#) will show the reader that the robot "wiggles" backward to deal with a frontal obstacle, i.e. the Functions RightBack and LeftBack were alternatively called. **This effect also explains why a DelayTime=0 would not work for the robot, as it would be switching so fast between the Functions "RightBack", "LeftBack" and "Forward" (within**

the Endless Loop) such that the robot looks “paralyzed” from the outside. By the way, the same thing happens to us humans when we multitask too much as nothing then gets achieved!

3.5 “Follower” Maneuvers using Left and Right IR Sensors

To round out the reader’s experience with using the Left and Right IR Sensors, let’s apply a “**reverse logic**” on them, essentially let’s make the robot follow an object when it detects something in front of it.

Let’s first learn how to track a moving object, when it is detected. The corresponding condition/action list can be as follows:

If there is no object detected, the robot should stay still (default behavior).

If there is an object detected by the Right IR Sensor, the robot should swing right.

If there is an object detected by the Left IR Sensor, the robot should swing left.

Should we also plan for the case when an object is detected and located right in front of the robot? Should it “stop” or should it show that it already “locks in” on the object in some other way? From the preceding Avoider coding work, we should be able to let the robot bounce between its two actions of swinging left and right to show that it got the object in its sight, right?

The program “Follower-1.tskx” was derived from the previous discussions using LOOP WHILEs. The algorithm used is shown in Fig. 3.23:

There was no discernable difference in performance between these two solutions that the author could see. In theory, “Follower-3.tskx” should be “faster” because it accesses the sensor ports only twice per each iteration of the overall Endless Loop, i.e. it uses less resources from the controller as compared to “Follower-2.tskx” which uses from 4 to 8 accesses to the sensor ports per each iteration of its overall Endless Loop. However, “Follower-2.tskx” can be more “accurate” in its actions as it always uses “fresh” data inputs (i.e. “there is no free lunch”).

Recapping Sections 3.4 and 3.5, the author’s goal is to show that programming “autonomous” behaviors is fun and challenging at the same time! Starting from the (now familiar) condition/action analysis, **three techniques can be used to create TASK codes and after some initial actions, they all used an outermost ENDLESS LOOP for the main algorithm:**

1) In Technique 1, for each iteration of the Endless Loop, latest readings from the used sensors were first recorded, then an IF-ELSE-IF structure was used to represent all possible but mutually exclusive condition/action pairs that corresponded to robot behaviors required to fulfill a given challenge. Example programs using Technique 1 are “IR_based_Maneuvers.tskx”, “IR_based_Maneuvers_App.tskx”, “Avoider-1.tskx”, “Avoider-2.tskx”, and “Follower-3.tskx”.

2) In Technique 2, the sensors reading task is still required first, but the IF-ELSE-IF structure is replaced by “parallel” IFs. This technique requires a deeper analysis of the original condition/action list to identify a “default” behavior, and other condition/action pairs that can work “on their own” or “in concert”. This technique also requires a short time delay incorporated into each robot action. Compared to Technique 1, Technique 2 usually results in a smaller number of statements used in the TASK code, meaning that the controller would take less time to do each iteration of the Endless Loop, and thus the robot may become more responsive to changes in the “World” (review Fig. 1.10). Example programs using this technique are “Avoider-3.tskx” and “Follower-1-IFs.tskx”.

3) Technique 3 is a variation of Technique 2 and it uses “parallel” LOOP WHILEs instead of “parallel” IFs. It combines “Data Acquisition” tasks in its Conditional Statements, thus it uses the least number of statements, as compared to the other two Techniques. Example programs using Technique 3 are “Avoider-4c.tskx” and “Follower-1.tskx”. “Follower-2.tskx” is a hybrid solution combining Techniques 3 and 1.

3.6 Remote Control Programming Techniques

Back in Section 1.1, “Instruction” and “Status” packets were conceptually described, and “RemoCon” packets were first mentioned. In this Section 3.6, a more detailed description of “RemoCon” packets would be provided, along with programming examples of different schemes for Remote Control of the “Spinning Top” robot:

- 1) Remote Control via the Virtual RC-100 Controller (from inside the TASK tool)
- 2) Remote Control using Touch Areas on a mobile device display via the R+m.PLAY700 App.
- 3) Remote Control using Speech Recognition via the R+m.PLAY700 App.
- 4) Remote Control using Tilt Sensor on a mobile device via the R+m.PLAY700 App.

3.6.1 Structure of a “Remocon” packet

A full “Remocon” packet is based on a 16-bit user-designed message wrapped up in a 6-byte packet (http://support.robotis.com/en/product/auxdevice/communication/rc100_manual.htm). As this book is for younger beginning programmers, we’ll restrict ourselves to using only the lower 10 bits of this 16-bit message. Advanced readers should consult Chapter 7 of Thai (2017) for a more complete treatment of this topic.

Both Physical and Virtual Remote Controller, RC-100 (respectively, Figs 1.5 and 1.6) were designed to behave programmatically in the same manner:

Each of the 10 buttons U-D-L-R-1-2-3-4-5-6 is represented by 1 bit in a 10-bit number/message, with the “U” button using the bit-0 position and the “6” button using the bit-9 position (see Fig. 3.26).

| RC-100 10-bit Message | | | | | | | | | | |
|-----------------------|---|---|---|---|---|---|---|---|---|---|
| Bit Position | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Button | 6 | 5 | 4 | 3 | 2 | 1 | R | L | D | U |
| Example Data | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |

Fig. 3.26 Structure of an RC-100 10-bit message.

AND Operation with Mask U-D-L-R

| | | | | | | | | | | |
|-------------------|---|---|---|---|---|---|---|---|---|---|
| Bit Position | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Button | 6 | 5 | 4 | 3 | 2 | 1 | R | L | D | U |
| Example Data | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| | | | | | | | & | | | |
| Mask U+D+L+R | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| Result from & Op. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

| A | B | A & B |
|---|---|-------|
| 1 | 0 | 0 |
| 1 | 1 | 1 |
| 0 | 0 | 0 |
| 0 | 1 | 0 |

Fig. 3.29 Result from ANDing operation between “Example Data” and Mask (U+D+L+R).

3.6.2 Remote Control of “Spinning Top” robot using Virtual RC-100

We are now ready to expand the concepts learned from Section 3.6.1 to work on a Remote- Control project for a CarBot which allows the user to use multiple Direction Buttons among the group U-D-L-R and to additionally set Low-Speed or High-Speed parameters on the fly using Buttons 1 and 3.

Fig. 3.33 Parallel IFs used for setting Direction(s) and Speed in “RC-MultiDirectionsSpeeds.tskx”.

3.6.3 Remote Control of “Spinning Top” robot using Touch Areas on Mobile Devices

This Section 3.6.3 shows how to write a TASK program that can be interfaced with a mobile device so that its “Touch Area” service can be utilized to perform Remote Control of a CarBot.

Let’s first look at a basic application for using the Touch Area features offered by the R+m.PLAY700 mobile App. This mobile App divides the display area in a 5x5 grid with a total of 25 touch zones (see Fig. 3.34). This App can monitor up to 5 Touch Areas simultaneously.

The program “MonitorTouchAreas.tskx” shows how to detect where the user had touched the mobile display and how to draw either a circle or a square in those detected zones.

First, we need to create a “Remote Control” project on the mobile device via the App R+m.PLAY700 so that we can assign 3 Shape Items (see Fig. 3.35 and part 1 of [Video 3.22](#)).

Fig. 3.36 shows the initial actions of the program “MonitorTouchAreas.tskx”:

- 1) Set the mobile display to the Portrait Mode (line 6).
- 2) Clear the display of all previous shapes – for just in case (line 7).
- 3) Wait for the user to whistle into the microphone of the mobile device (line 9) before going into the main Endless Loop (line 11 of Fig. 3.37).

```

3  START PROGRAM
4  {
5  // Getting remote device ready using SMART commands
6  SMART: Screen Rotation = Portrait Mode (1)
7  SMART: Shapes Display = 0
8  // User needs to make sure that robot is connected to mobile device. User whistles loudly when ready.
9  WAIT WHILE ( SMART: Noise (dB) < 50 )

```

Fig. 3.36 Initial actions by program “MonitorTouchAreas.tskx”.

```

11  ENDLESS LOOP
12  {
13  Touch1 = SMART: Touch Area 1
14  Touch2 = SMART: Touch Area 2
15  // Clearing screen display
16  SMART: Shapes Display = 0
17  IF ( Touch1 != 0 )
18  {
19  SMART: Shapes Display = [Position:Touch1],[Item:1],[Size:60],[Color:White]
20  High-resolution Timer = 0.250sec
21  WAIT WHILE ( High-resolution Timer > 0 )
22  }
23  IF ( Touch2 != 0 )
24  {
25  SMART: Shapes Display = [Position:Touch2],[Item:2],[Size:60],[Color:Green]
26  High-resolution Timer = 0.250sec
27  WAIT WHILE ( High-resolution Timer > 0 )
28  }
29  }

```

Fig. 3.37 Main Algorithm in program “MonitorTouchAreas.tskx”.

Fig. 3.37 describes the main algorithm controlled by a familiar Endless Loop, with the following actions for each iteration of this loop:

- 1) in Lines 13 and 14, SMART commands are used to access the Touch Area 1 and 2 where the user may have touched the mobile display. If no touching is done by the user, a zero value is saved into Parameters “Touch1” and “Touch2”, otherwise a number between 1 and 25 will be returned from those SMART

commands. The “first” touch by the user will be recorded into “Touch1” and the “second” touch will be recorded into “Touch2”.

- 2) The display screen is cleared of previous graphics (line 16).
- 3) Two parallel IFs are used, one for each Touch, as 1 or 2 touches can be performed by the user at any time. If “Touch1” is non-zero (line 17), then a Circle (i.e. Shape Item 1) will be displayed at the location defined by the value contained in “Touch1” in White color (line 19). Please see Fig. 3.38 for details of the RHS of the LOAD statement at line 19 of Fig. 3.37.

As we can only use up 2 Touch Areas simultaneously for this RC project, we cannot achieve the multi-button features allowed by the RC-100 as shown previously in Section 3.6.1. However, we still want to be able to set the CarBot’s maneuver directions independently from its Speed setting. The solution approach is to use one “Touch” for the main maneuver direction and the other “Touch” for a secondary maneuver direction or for setting the Speed value.

The program “RC-TouchAreasDirectionSpeed.tskx” describes how this solution approach is implemented:

- 1) The initial actions are quite extensive in this solution (see Fig. 3.39). Three parameters are used “Speed” (= 256), “TimeDelay1” (= 100 ms) and “TimeDelay2” (= 25 ms) as shown in lines 5-7. Next the mobile device is set to Portrait mode, and its display cleared, then it waits on the user to whistle into it (line 12) to get the CM-50 to finish up its main actions.
- 2) Next, six Text Items are put at the appropriate display locations so that they can be used as icons standing in for maneuver directions and speed settings (see Fig. 3.40 and lines 15-21 in Fig. 3.39). Essentially, these six Text Items (and their underlying Touch Zones) serve as the user interface for the program “RC-TouchAreasDirectionSpeed.tskx”, similar in function to the Virtual RC-100 user interface.

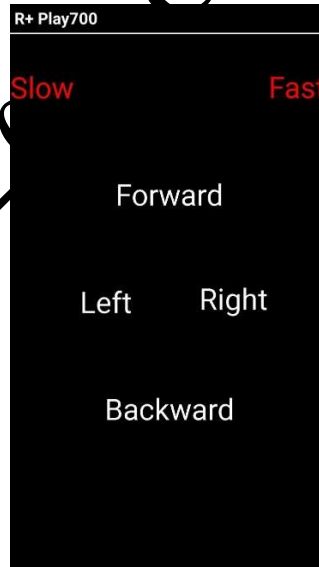


Fig. 3.40 Screen capture showing user interface in “RC-TouchAreasDirectionSpeed.tskx”.

3.6.4 Voice Control of “Spinning Top” robot using R+m.PLAY700 App

This project explores the feasibility and performance of a Voice Control scheme to remotely control a CarBot to perform basic maneuvers. Back in Section 3.3.2, Text Items were useful to display texts and could serve as “inputs” to the Text-to-Speech engine of the R+m.PLAY700 App. The same Text Items can also be used as “outputs” from this App’s Speech Recognition engine which was based on the Google Speech API (<https://cloud.google.com/speech/>), meaning that your mobile device needs to be connected to the web in order to use this feature and consequently this process needs “seconds” to perform, not “milliseconds” like with the RC-100.

Another important observation also needs to be made regarding the difference in Remote Control “technologies” that are used so far. When using the RC-100 module, potentially our RC scheme could rely upon 10 simultaneous Input Sources as represented by the 10 Buttons “UDLR123456”. When using the Touch Areas of a mobile display, we can only rely on two simultaneous Input Sources, Touch Areas 1 and 2 even though the potential number of touch zones is 25. In this Section 3.6.3, we plan to use Speech Recognition technology which potentially can recognize human verbal inputs, but it must be noted that we will be restricted to ONE verbal input stream at any one time, as humans do not know how to multiplex their speech pattern yet!

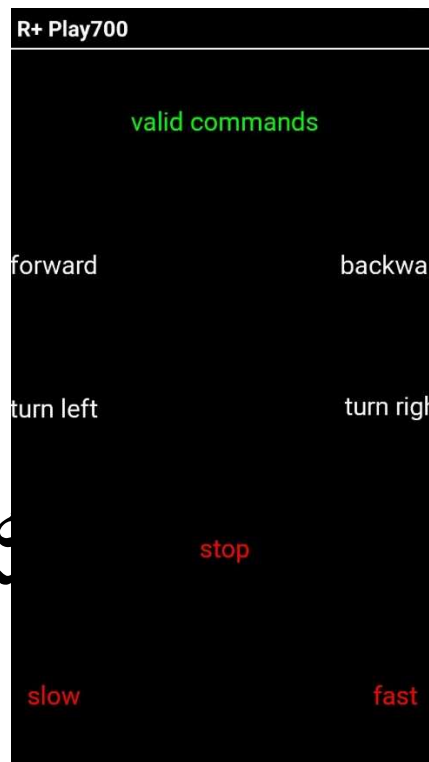


Fig. 3.45 Menu of Valid Verbal Commands in program “RC-UsingSpeech.tsx”.

Fig. 3.46 describes the Main Algorithm which uses nested ENDLESS LOOPS:

- 1) The Outer ENDLESS LOOP is entered at Line 13. The “Whistle” label (line 15) is an important “Jump To” location that will be explained later with the Inner ENDLESS LOOP.
- 2) Line 16 shows that the user needs to whistle again to make the mobile device clear its display (line 17) and next voice out and display Text Item 1 (i.e. “ready”) via lines 17-20.

3) Lines 21-22 shows that the robot would wait 2 seconds for the user to get ready to verbalize his or her command.

4) The Inner ENDLESS LOOP is entered at line 24 and the Voice Recognition task gets started by line 27 followed by a WAIT WHILE for this process to finish (line 29). When line 27 is executed the mobile device displays “Speak to Me” (see Fig. 3.47) and waits for the user to verbalize the command (within a given time-out period). When the user is finished with verbalizing and while line 29 is being executed, it would display “Recognizing” (see Fig. 3.48).



Fig. 3.47 “Speak to Me” display in program “RC-UsingSpeech.tskx”.



Fig. 3.48 “Recognizing” display in program “RC-UsingSpeech.tskx”.

5) Next, the Result of Speech Recognition is saved in Parameter “VoiceCommand” (line 30). It is a number, between 1 and 10, corresponding to the index of the “recognized” Text Item as shown in Fig. 3.43.

6) The first IF construct (lines 31-35) checks to see if the value saved in “VoiceCommand” is valid (i.e. different from “-1”). If it is valid, the Function “ProcessCommand” is called (line 33) and when this function is done, the parameter Result of Speech Recognition is reset to zero (line 34).

7) Function “ProcessCommand” (see Figs. 3.49 and 3.50) starts by checking if “VoiceCommand” is non-zero, i.e. an actual match is made with one of the Text/Voice Recognition Items shown in Fig. 3.43.

8) Next, the IF-ELSE-IF construct (Statements 107 to 131) is used to trigger the appropriate robot maneuvers (Forward, Backward, Turn Right, Turn Left or Stop) depending on the value found in “VoiceCommand” at runtime. Similarly, “VoiceCommand” is used to figure out the correct Slow and Fast Speed settings to use (lines 132 to 143 in Fig. 3.50).

```

103 FUNCTION ProcessCommand
104 {
105 IF ( VoiceCommand != 0 )
106 {
107 IF ( VoiceCommand == Text Item 2 )
108 {
109 CALL Forward
110 SMART: Text Display = [Position:(3,3)],[Item:2],[Size:100],[Color:White]
111 }
112 ELSE IF ( VoiceCommand == Text Item 3 )
113 {
114 CALL Backward
115 SMART: Text Display = [Position:(3,3)],[Item:3],[Size:100],[Color:White]
116 }
117 ELSE IF ( VoiceCommand == Text Item 4 )
118 {
119 CALL TurnRight
120 SMART: Text Display = [Position:(3,3)],[Item:4],[Size:100],[Color:White]
121 }
122 ELSE IF ( VoiceCommand == Text Item 5 )
123 {
124 CALL TurnLeft
125 SMART: Text Display = [Position:(3,3)],[Item:5],[Size:100],[Color:White]
126 }
127 ELSE IF ( VoiceCommand == Text Item 6 )
128 {
129 CALL Stop
130 SMART: Text Display = [Position:(3,3)],[Item:6],[Size:100],[Color:White]
131 }

```

Fig. 3.49 Part 1 of Function “ProcessCommand” in program “RC-UsingSpeech.tsx”.

```

132 ELSE IF ( VoiceCommand == Text Item 9 )
133 {
134 Speed = 200
135 TurnSpeed = 160
136 SMART: Text Display = [Position:(3,3)],[Item:9],[Size:100],[Color:White]
137 }
138 ELSE IF ( VoiceCommand == Text Item 10 )
139 {
140 Speed = 300
141 TurnSpeed = 200
142 SMART: Text Display = [Position:(3,3)],[Item:10],[Size:100],[Color:White]
143 }
144 }
145 ELSE IF ( VoiceCommand == 0 )
146 {
147 CALL Stop
148 InvalidCommand = TRUE (1)
149 SMART: Text Display = 0
150 SMART: Text to Speech (TTS) = Text Item 7
151 SMART: Text Display = [Position:(3,3)],[Item:7],[Size:100],[Color:Red]
152 WAIT WHILE ( SMART: Text to Speech (TTS) != 0 )
153 CALL Menu

```

Fig. 3.50 Part 2 of Function “ProcessCommand” in program “RC-UsingSpeech.tsx”.

9) If no match is found for the user’s verbal command, “VoiceCommand” would have a zero value and this “Invalid Command” case is dealt with by the construct comprising lines 145 to 153 (see Fig. 3.50).

The robot is then stopped (Statement 147) and “InvalidCommand” is set to TRUE (line 148). Next, a Display and Text-to-Speech procedure is performed with Text Item 7 (i.e. “Invalid Command” – lines 149-152). Lastly, the Menu of Fig. 3.45 is redisplayed (line 153) and the program flow control returns to the main program at line 34 which goes on to line 36 (see Fig. 3.46).

10) Next, if “InvalidCommand” happened to be set to TRUE from inside the Function “ProcessCommand”, then “InvalidCommand” was reset to FALSE (line 38), and the program flow control JUMPs (line 39) to LABEL “Whistle” (line 15) to restart the whole algorithm all over again.

3.6.5 Remote Control using Gradient Tool (Tilt Sensor) on Mobile Device

In this sub-section, we’ll use the Gradient Tool provided by the R+m.PLAY700 App (see Fig. 2.3, under the “Sensor” group). This Gradient Tool uses the accelerometer and magnetometer sensors usually built-in on most mobile devices, however some entry-level devices may not be equipped with such sensors, thus this tool may not work properly for some mobile devices that the reader may use. It is recommended that readers check this functionality on their mobile devices first (see [Video 3.25](#)). Once the Gradient Tool (Tilt Sensor) is accessed via the “Editing” of a SMART Project within the R+m.PLAY700 App, the mobile device would display a screen resembling the one shown in Fig. 3.51. In this instance, the mobile device was tilted such that its Front Edge was UP (18 degrees) and its Right Edge was also UP (28 degrees), within respect to the horizontal plane. Readers should notice that “negative” gradients are not “reported” by this tool.

For this application “RC-Tilt-Sensor.tsx”, the **Tilt Sensor is used as a joy-stick controller**, thus the nomenclature of the sensing parameters and their use in various conditional statements would read, at places, **like a complete reversal of the logic used with the RC-100 controller!**

```

11 | ENDLESS LOOP
12 | {
13 |   SMART: Screen Rotation = Landscape Mode (2)
14 |   Gradient_Up = SMART: Acceleration Sensor (U)
15 |   Gradient_Down = SMART: Acceleration Sensor (D)
16 |   Gradient_Left = SMART: Acceleration Sensor (L)
17 |   Gradient_Right = SMART: Acceleration Sensor (R)
18 |   SMART: Number Display = [Position:(3,2),[Item:Gradient_Up],[Size:100],[Color:White]
19 |   SMART: Number Display = [Position:(2,3),[Item:Gradient_Left],[Size:100],[Color:White]
20 |   SMART: Number Display = [Position:(4,3),[Item:Gradient_Right],[Size:100],[Color:White]
21 |   SMART: Number Display = [Position:(3,4),[Item:Gradient_Down],[Size:100],[Color:White]
22 |   IF ( Gradient_Up < 5 && Gradient_Down < 5 && Gradient_Left < 5 && Gradient_Right < 5 )
23 |   {
24 |     CALL Stop
25 |   }

```

Fig. 3.53 Part 1 of Main Algorithm for “RC-Tilt-Sensor.tsx”.

Fig. 3.53 shows Part 1 of the Main Algorithm (with the familiar EndLess Loop) whereas:

1) This Endless Loop starts by constraining the mobile device into a landscape display mode (line 13). This action is needed to prevent the mobile device from switching, unintentionally, between its two display modes (portrait and landscape) later during runtime of this program when this device is manipulated

as a joy-stick, as this would confuse the user considerably. This is a Human-Machine-Interface issue more than a Robotics one. The reader is encouraged to comment out Line 13 to see its effects at runtime.

2) The mobile device's reference plane is the horizontal plane. Thus all 4 "Gradients or Accelerations" parameters (in degrees) would be reported as 0 degree when the device is positioned flat in a horizontal manner. Furthermore, the device would only provide POSITIVE Gradient values whenever they were ABOVE the horizontal plane. For example, if the Front edge of the phone is above the horizon, the Gradient_Up parameter would show a positive value while the Gradient_Down parameter would show a "zero" value (although logically it should have been a "negative" value and equal in magnitude to Gradient_Up). Lines 14-17 record the U-D-L-R gradients and lines 18-21 display them on the mobile display in a cross configuration centered on the screen.

3) The author wants the robot to be still when the mobile device is held "mostly" in a horizontal plane thus the use of the "5-degrees" as strict upper limits in line 22 of Fig. 3.53.

3.7 Autonomous Line Tracking & Obstacle Avoidance with NIR Sensors

For this Section 3.7, we will be using a CarBot configuration as described in Fig. 3.55. The goal is to program this CarBot to go forward and follow an arbitrary track created using standard black electrical tape laid out on brown Craft paper. This CarBot would have to use its Left and Right NIR sensors to keep it "on-track". During its progress along this track, if the CarBot finds a frontal obstacle, it should be able to swing around, re-discover the black track and follow it again but in the opposite direction.

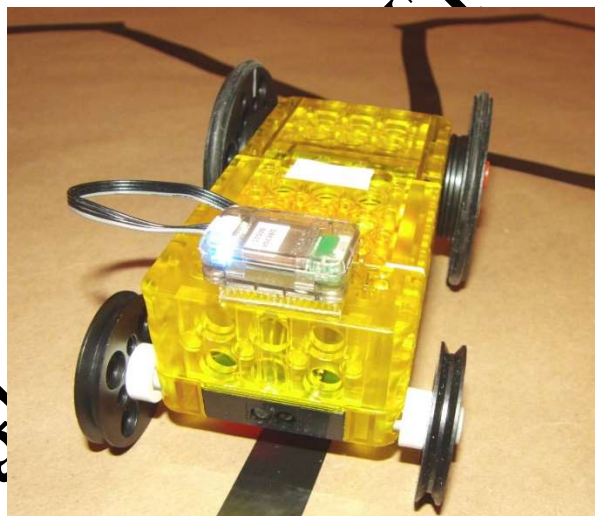


Fig. 3.55 CarBot used as Autonomous Line Tracker & Obstacle Avoider.

Let's first analyze how the three NIR Sensors behave when the CarBot is laid on the black track under the four main conditions that would occur in this challenge: a) when the track is under the bot's Left IR Sensor; b) when the bot is centered on the track; c) when the track is under the bot's Right IR Sensor; and d) when the bot is gone completely off-track (see Fig. 3.56).



Fig. 3.56 Values for L-R-C IR Sensors for 3 configurations between CarBot and Black Track.

In Fig. 3.56 and going from Column (A) to Column (C) to check for the readings from the Right IR Sensor (Address 91): one can see that its value is around 740-750 whenever it sees the background Craft paper, and that its value is reduced to around 340 when this sensor is “seeing” the black track. Similarly, for the Left IR Sensor (Address 93), it reads around 400 when it “sees” the black track and between 640 and 670 when it “sees” the Craft paper surface. Most importantly, Column (B) shows that the spacing between the Left and Right IR Sensors is larger than the width of the black electrical tape as both sensors read “high” here (e.g. 744 and 646) - in other words, both sensors are only seeing the brown paper. **Thus, it is physically impossible for this CarBot to figure out via its Left and Right NIR sensors whether it is centered on the black track (case b) or that it is completely off-track (case d)!**

The previous experimental analysis shows that there are only two conditions (mutually exclusive) that can be monitored unequivocally by this CarBot hardware:

- 1) If the Left IR Sensor can see the black track then it should turn left in its attempt to re-align the bot with the black track.
- 2) If the Right IR Sensor can see the black track then it should turn right in its attempt to re-align the bot with the black track.

Otherwise, the robot really cannot be “sure” of anything else, and so it should continue its default maneuver (i.e. going forward). From previous sections, we are then expecting to see an IF-ELSE-IF structure for three mutually exclusive alternatives in the programming solution for this challenge.

The Function “SwingAround” (Fig. 3.59) describes a combination of “timed” and “conditional” right turns to accommodate for the fact that the “obstacle” can be found at any spot on the black track:

First, a 0.125s timed right-turn is used (lines 82-85) because at this point in time, the CarBot is mostly lined up with the black track and it just needs to be nudged a bit to the right to make sure that Right IR Sensor is now “seeing” the Craft paper (i.e. “white” - the condition $(IR_Right > LineDistance)$ is TRUE).

```

79 FUNCTION SwingAround
80 {
81 // Turn right for 0.125 second to clear IR-Right sensor off the track
82 PORT[1]:Geared Motor = CCW:0 (0.00%) + 250
83 PORT[2]:Geared Motor = CCW:0 (0.00%) + 250
84 High-resolution Timer = 0.125sec
85 WAIT WHILE ( High-resolution Timer > 0.000sec )
86
87 // Keep on turning right as long as IR-Right sees white
88 WAIT WHILE ( IR Right > LineDistance )
89
90 // IR-Right now sees Black, but OK to keep on turning right until IR-Right sees White again
91 WAIT WHILE ( IR Right <= LineDistance )
92 // Bot should be lining up with the track but in the reverse direction at this point
93 }

```

Fig. 3.59 Function “SwingAround” for program “LineFollower_ObstacleAvoider.tskx”.

Next, the reader needs to recall that the previous commands for the CarBot to turn right (lines 82-83) are still in effect. The 2 WHILE loops (lines 88 and 91) are “conditional” loops so that the CarBot can adjust to “any” section of the arbitrary track where the “SwingAround” maneuver may be needed. These 2 loops are used to ensure that an orderly process is followed to detect whether the Right IR Sensor is seeing “black” or “white” (e.g. brown Craft paper):

The first loop (line 88) is active whenever Right IR sees “white”, i.e. whenever the CarBot has **not** yet crossed back into the black track. When this situation happens, i.e. when IR Right sees “black”, the first loop’s conditional expression becomes false and this loop is exited, and the second loop is entered (line 91).

The purpose for the second loop (line 91) is to keep the CarBot turning right some more until IR Right sees “white” (i.e. it has just passed the other edge of the black track). At this point in time, the CarBot should be mostly lined up with the black track (but in the reverse direction), thus this Function is exited and another iteration of the Main Endless Loop is executed, resulting in the Carbot’s default maneuver of Going Forward.

3.8 Using Mobile Video Camera for Color Line Tracking

In this section, we’ll use the Line Detection tool provided by the R+m.PLAY700 App (see Fig. 2.3, under the “Vision” group). This tool allows some rudimentary “real-time” image processing functions on the mobile device that are however accessible to the CM-50 via TASK programming. Fig. 3.60 displays the bottom view of a CarBot with a Smartphone mounted on top, showing the camera view port and the built-in LED light source that can be used to “light up” the viewed scene.



Fig. 3.60 Bottom View of a CarBot equipped with a Smartphone.

The R+m.PLAY700 App divides the phone camera's entire sensing area into 25 equal zones, i.e. 5 rows by 5 columns, just like how it partitions its 25 Display Areas for Texts, Shapes and Numbers, and also for its 25 Touch Areas (see Fig. 3.34). [Video 3.28](#) shows how to check the functionality of this tool using a piece of white paper with different color lines on it. Fig. 3.61 is a screen capture of the mobile device while it is supposed to track a Red Line. In this figure, the reader can see that this tool classifies a Yellow Line erroneously as a second Red Line (see the little window in Upper Left corner of Fig. 3.61) – this is the limitation of the current R+m.PLAY700 App.

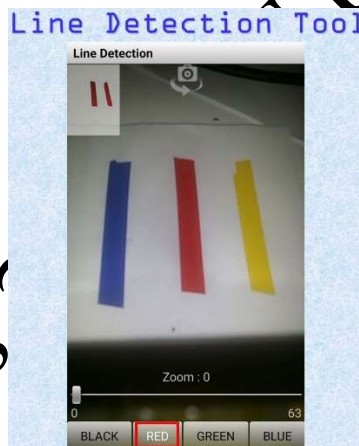


Fig. 3.61 Performance of Line Detection Tool for Red and Yellow lines.

The current Line Detection tool uses only the top row of the sensing zones (i.e. zones 1-5), mainly because of the limitation in throughput for the Bluetooth communication channel which would be overwhelmed when data from all 25 zones were streamed back to the CM-50 robot to process. When the CarBot as configured in Fig. 3.61 is laid on top of a Red line, Fig. 3.62 shows two important features to be considered when using this tool inside a TASK program:

- 1) The image captured is out of focus because the Red line turns out to be set too close to the camera, but this will have no effect in this particular application where we only care to know about which zone, among zones 1 through 5, contains the Red line (i.e. a fuzzy red blob in left image in Fig. 3.62).
- 2) “Zoom” may have to be set so that the top row of the camera sensor (i.e. zones 1 to 5) can detect enough of the fuzzy Red line so that it can display a specific zone number (which is “3” in Fig. 3.62)

where it has found a “red blob”. “Zoom” level is set to “4” in Fig. 3.62. Please note that this “issue” is due to the LED lighting condition used by the author, i.e. a bit too dark at the top edge of the camera sensor in left image and zooming takes care of this problem as shown in right image. If the reader uses ambient diffuse light, this issue may not arise for the reader at all – so be sure to check your actual usage situation to determine whether “zooming” is needed or not (see Fig. 3.63 where zoom was not needed and the detected zone was found to be 2).

The Color Line Detection process has 3 general steps:

- 1) Set SMART DEVICE “Camera Sensor” to “Line Detection Mode”.
- 2) Set SMART DEVICE “Vision – Tracking Color” to a chosen color among 4 pre-defined “Colors”: Black (=2), Red (=3), Green (=4) and Blue (=5).
- 3) Read SMART DEVICE “Vision – Line Detection Area” to obtain a numerical value which would be between 1 and 5 when the “assigned” color is found, otherwise it would return a zero value.

The solution TASK program to this challenge is called “ColorLineFollowTaskx” and it uses Nested Loops: i.e. an external ENDLESS LOOP along with an internal LOOP WHILE (depending on the detected zone number).

3.9 Multimedia Moody Dog with R+m.PLAY700

This project illustrates the use of graphics display and audio files on the mobile device and shows how to integrate their timing with sensors on board the CM-50.

The corresponding TASK code is called “MoodyDogBot.tskx”. It uses two Audio files: “Small-dog-barking.mp3” and “Angry-dog.mp3”. It uses 4 background images: “DogBot.png”, “MoodyFella.png”, “FriendlyDog.png” and “AngryDog.png”. The reader will need to create a Custom Project inside the R+m.PLAY700 App and these 6 files need to be stored in specific folders of this “Moody Dog” SMART project (see Figs. 3.66 and 3.67). [Video 3.30](#) demonstrates how to set up this project on a mobile device).

| Audio Playback | | Add |
|----------------|-----------------------|-----|
| 1 | Small-dog-barking.mp3 | |
| 2 | Angry-dog.mp3 | |

Fig. 3.66 Audio Files used by Moody Dog SMART Project.




| | Background | Add |
|---|---|-----|
| 1 |  DogBot.png | |
| 2 |  MoodyFella.png | |
| 3 |  FriendlyDog.png | |
| 4 |  AngryDog.png | |

Fig. 3.67 Background Graphics Files used by Moody Dog SMART Project

Fig 3.68 shows the Initial Actions (Part 1) of the program “MoodyDogBot.tskx”:

First, the CM-50 plays Melody 2 (lines 5-7) and waits for the user to whistle into the mobile device (line 9).

```

4 // Initial Actions
5 Buzzer Timer = Melody Time
6 Buzzer Index = Melody No.2 (2)
7 WAIT WHILE ( Buzzer Timer > 0.0sec )
8 // Waiting for user to whistle into mobile device when ready, then displays Background Image 1
9 WAIT WHILE ( SMART: Noise (dB) < 50 )
10 SMART: Background Image = 1
11 // Announce and Display "Ready!"
12 SMART: Text to Speech (TTS) = Text Item 1
13 SMART: Text Display = [Position:(3,3)],[Item:1],[Size:200],[Color:Green]
14 WAIT WHILE ( SMART: Text to Speech (TTS) != 0 )
15 High-resolution Timer = 2.000sec
16 WAIT WHILE ( High-resolution Timer > 0.0sec )
17 SMART: Text Display = 0

```

Fig. 3.68 Part 1: Initial Actions in “MoodyDogBot.tskx”

Next, the mobile device displays Background Image 1 (line 10). It also announces and displays Text Item 1 “Ready!” (lines 12-14).

The CM-50 then waits for 2 seconds (lines 15-16), and sends a Text Display command telling the mobile device to clear its Text Screen (line 71) to get ready for Part 2 of the algorithm.

Part 2 of the algorithm (see Fig. 3.69) is an Endless Loop allowing the user to use hand claps to select between the “Friendly” (1 clap) and “Angry” (2 claps) moods of the DogBot:

The Background Image 2 “Moody Fella~” is displayed (line 22), and the CM-50 waits for the user to clap once for the “Friendly Dog” mood and twice for the “Angry Dog” mood (line 23-25).

Next, depending on the number of claps recorded (1 or 2, saved in Parameter Claps), one of the two parallel IFs constructs would be triggered (lines 27-30 and 32-35), effectively CALLing upon either Function “Friendly” or Function “Angry”. Both Functions share the same code design. After these functions are executed, a new iteration of the EndLess Loop begins anew.

3.10 Using Musical Instruments on CM-50 and Mobile Device

When using the CM-50 in concert with the R+m.PLAY700 running on a mobile device, there are many audio output devices that can be activated at the same time:

The Buzzer on the CM-50, providing musical scales and melodies (see Section 3.3.1).

Two simultaneous audio sources on the mobile device, capable of playing most audio files (see Section 3.9).

Musical scales and Octave levels with One Musical Instrument chosen among 128 possible Instruments (more details later in this Section 3.10).

First, let's examine the structure of the SMART CONSTANT that can be used to play a SMART Musical Instrument. This SMART CONSTANT has 4 bytes, but only the lower 3 bytes are used.

The first (lowest) byte is kept for MUSICAL SCALE or NOTE (possible values are 1 to 12): e.g. "1" for "Do", "2" for "Do#", etc... with "12" corresponding to "Shi".

The second byte is kept for OCTAVE (possible values are 1 through 10).

The third byte is kept for INSTRUMENT TYPE (128 possible types, supported on Android devices only): "1" for "Acoustic Grand Piano" and "128" to "Gunshot", with other instruments in between.

In this application, the goals are to make the CM-50's buzzer cycle through its Melody Index from 0 to 24, and to make the mobile device cycle through its Musical Instrument Type from 1 to 128. For the two "mobile" audio sources, the plan is to let each audio source play its own audio file continuously.

The program "MusicalInstruments.tsx" shows how to set up needed parameters and how to activate these 4 audio sources within an Endless Loop. Fig. 3.71 describes the Initial Actions:

3.11 More Applications with Mobile Camera: Face and Intruder Detections

The R+m.PLAY700 App has a tool for finding faces, well to be exact, it finds two blobs that can be construed as a pair of eyes! The program "FaceDetection.tsx" is a simple application of this tool (see Fig. 3.76):

The user starts the application by whistling into the mobile device (line 5).

Line 8 set the Mobile Device into a Portrait Mode.

Line 9 chooses the Back Camera.

Line 10 sets the Camera Sensor into the Face Detection Mode, and waits for the hardware to settle down by using the SMART TIMER to wait for 2 seconds (lines 11-12). Please note that the SMART TIMER resides on the Mobile Device, this timer is not the same as the CM-50's Hi-Res Timer that we had been using so far.

The Endless Loop (lines 13-23) contains the main Face Detection procedure:

Line 15 invokes the actual "Face Detection Area" software tool to run on the current image captured via the Back Camera. If among the searched 25 zones (see Fig. 3.34), it can find 2 blobs that are positioned in such a way that they can be classified as a pair of eyes, it returns that zone number which is then saved into Parameter DetectedZone. If DetectedZone has a zero value, this means that "no eyes" had been found.

Another R+m.PLAY700 tool for the video camera is the Motion Detection tool that can be applied to an Intruder Alert project. The solution program is “IntruderAlert.tskx” (see Fig. 3.78):

The user starts the application by whistling into the mobile device (line 5).

Line 8 set the Mobile Device into a Portrait Mode.

Line 9 chooses the Back Camera.

Line 10 sets the Camera Sensor into the Motion Detection Mode, and waits for the hardware to settle down by using the SMART TIMER to wait for 2 seconds (lines 11-12). Please again note that the SMART TIMER resides on the Mobile Device, this timer is not the same as the CM-50’s Hi-Res Timer that we had been using so far.

```
5 WAIT WHILE ( SMART: Noise (dB) < 50 )
6
7 // Setting Back Camera to Motion Detection Mode & wait for 2 seconds
8 SMART: Screen Rotation = Portrait Mode (1)
9 SMART: Camera Selection = Back Camera (1)
10 SMART: Camera Sensor = Motion Detection Mode (3)
11 SMART: Smart Timer = 2.0sec
12 WAIT WHILE ( SMART: Smart Timer > 0.000sec )
13
14 ENDLESS LOOP
15 {
16   DetectedZone = SMART: Motion Detection Area
17   IF ( DetectedZone != FALSE (0) )
18   {
19     SMART: Photo Capture = Capture (1)
20     Buzzer Timer = Melody Time
21     Buzzer Index = Melody No.20 (20)
22     WAIT WHILE ( Buzzer Timer > 0.000sec )
23     WAIT WHILE ( SMART: Photo Capture != 0 )
24   }
25 }
```

Fig. 3.78 TASK Code for Intruder Alert Application.

Copyrights 2022

Optics LLC

Chapter 4: Using R+SCRATCH & SCRATCH® 2

MIT's SCRATCH 2 software is a very popular software platform with millions of users worldwide (<https://scratch.mit.edu/>), and reference texts on this subject are too numerous to list in this chapter. The author would recommend such works as Ford (2014) and Warner (2015) for self-learners who want to have a thorough understanding of the SCRATCH 2 language. If the reader is a teacher/instructor then the author would also recommend the book by Bagge (2015) and his Code-IT web site (<http://code-it.co.uk/philbagge>). In this Chapter, only selected SCRATCH 2 features, most applicable to robotics, would be presented.

ROBOTIS designs the software tool R+SCRATCH to be a helper application that links the SCRATCH 2 IDE directly to the CM-50 Controller's Firmware (see Figs. 1.8 and 4.1). This means that any TASK code previously downloaded onto the CM-50 is disabled (not erased) as soon as R+SCRATCH connects to the CM-50 (see Part 1 of [Video 4.1](#)).

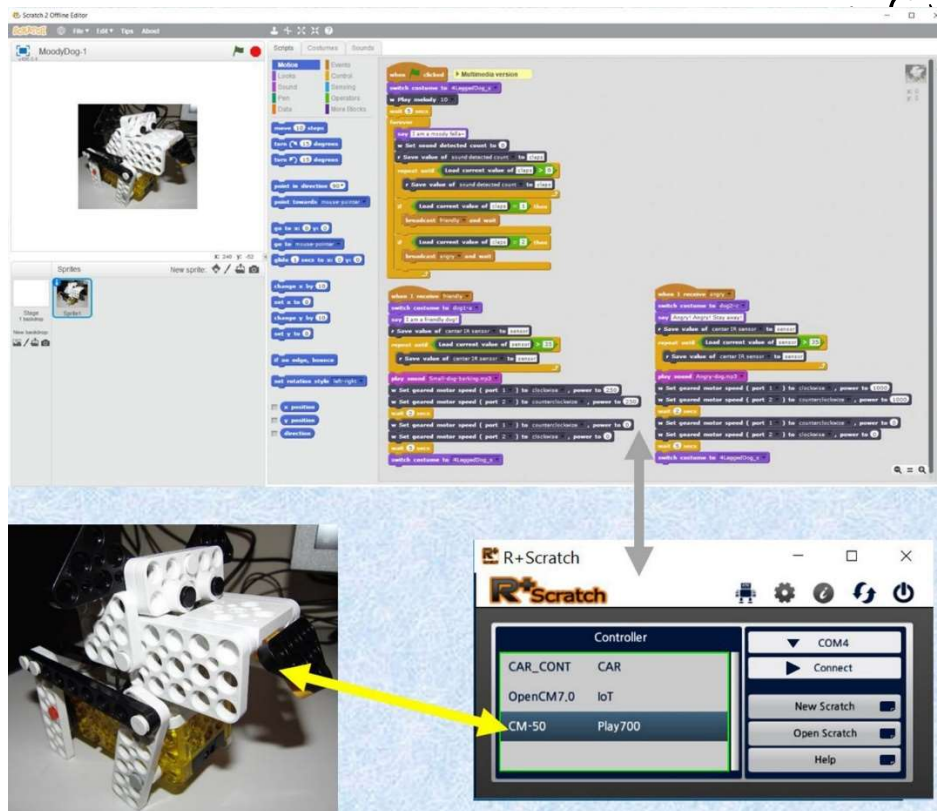


Fig. 4.1 Information Flow between SCRATCH 2, R+SCRATCH and CM-50 Controller.

ROBOTIS also creates a custom SCRATCH HTTP Extension tool (https://wiki.scratch.mit.edu/wiki/Scratch_Extension) to the SCRATCH 2 Off-Line Editor to make available 8 SCRATCH Block Codes that can interface with the CM-50's firmware (see Fig. 4.2). The three "Play" and the three "Set" Block Codes are of the "w" type, i.e. a "write" type of command. Each "write" command requires 50 ms delay to progress, from the SCRATCH 2 IDE through the R+SCRATCH helper application, and all the way down to the CM-50 Firmware and actuators mounted on the PLAY700 robot. There is only one "r" (i.e. "read") type of command, "Save value of", which has a pull-down menu to access two types of sound counter and the three built-in IR Sensors (Left, Center, Right). Each "read" command requires 70-100 ms delay to process, once it is triggered at the SCRATCH 2 IDE level. When using TASK codes as shown

in Chapter 3, these types of commands cost much less time to be performed (less than 5 ms), so the much longer time delays when using SCRATCH 2 create important and sometimes adverse side effects (programming and physical) affecting a robot runtime performance. These effects will be shown and discussed further in later parts of this Chapter.

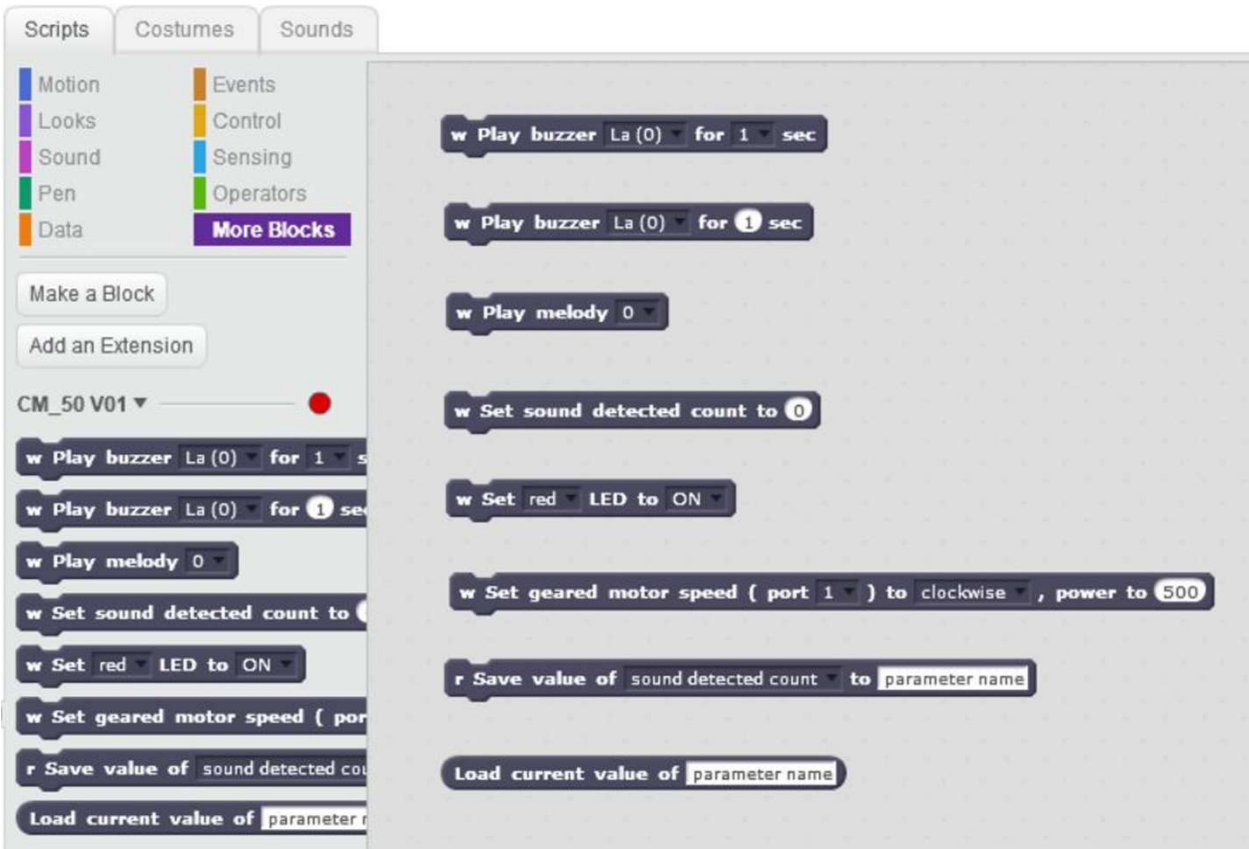


Fig. 4.2 ROBOTIS CM-50 Block Codes compatible with SCRATCH 2 IDE.

Part 2 of [Video 4.1](#) shows a basic usage of the tool chain R+SCRATCH/SCRATCH 2 Offline Editor and the “Spinning Top” robot (see Fig. 2.10), resulting in a project named “Maneuvers-1.sb2” to be described in Section 4.1 (see Fig. 4.3). Although SCRATCH 2 is designed for beginner programmers aged 8 or older, it has quite a sophisticated structure for its “project”:

Each SCRATCH project can have several actors called “SPRITES” and the “Maneuvers-1” project uses a single Sprite called “Sprite 1” (see Fig. 4.3 – bottom left panel). The large area on the top left panel, where a larger picture of Sprite 1 can be found, is called the STAGE.

Attached to each Sprite are SCRIPTS (actual codes), its COSTUMES (graphics components triggered by its SCRIPTS), and its SOUNDS (audio components also controlled by its SCRIPTS) – see top of mid-panels of Fig. 4.3.

4.1 Basic Robot Programming with SCRATCH

In this section, the goal is to show the similarities and differences in creating and running 2 pieces of codes on the same robot with the same required physical behavior (i.e. “go forward for 0.5 second”). One is a TASK

code from Section 3.2 named “Maneuvers-1.tskx” and the other is a SCRATCH program called “Maneuvers-1.sb2” (see [Video 4.3](#) for complete steps).

Fig. 4.4 displays the MAIN SCRIPT which starts when the user clicks on the GREEN FLAG block (which has similar functions as the START PROGRAM statement used in TASK codes):

The GREEN FLAG block can be found in the EVENTS menu located in the SCRIPTS tab.

Next, Variables “Speed” and “DelayTime” are set respectively to “512” and “0.5” (lines 1-2, i.e. same functionality as Assignment statements inside TASK). The Variable parameters can be created and set using items found in the DATA menu of the SCRIPTS tab.

Then, Port 1 Motor is set to a Clockwise motion with a power equaled to Variable Speed (line 4), while Port 2 Motor is set to a Counter-Clockwise motion with a power also equaled to Speed (line 5). Together, these two Code Blocks make the Bot go forward. Inside TASK codes, we would use an equivalent COMPUTE type of command here. These Code Blocks are part of the ROBOTIS EXTENSION so they can be found in the MORE BLOCKS menu of the SCRIPTS tab.

Line 6 just makes the PC (not the actual robot) wait for 0.5 second, because a SCRATCH program runs inside the PC’s memory and **not** on the CM-50’s memory like for a TASK code. The WAIT block can be found in the CONTROL menu of the SCRIPTS tab. This step is clearly easier to use than setting up the Hi-Res Timer with a delay of 500 ms in TASK codes as shown in Section 3.2.

Lastly, lines 7 and 8 turn off the motors to make the robot stop.

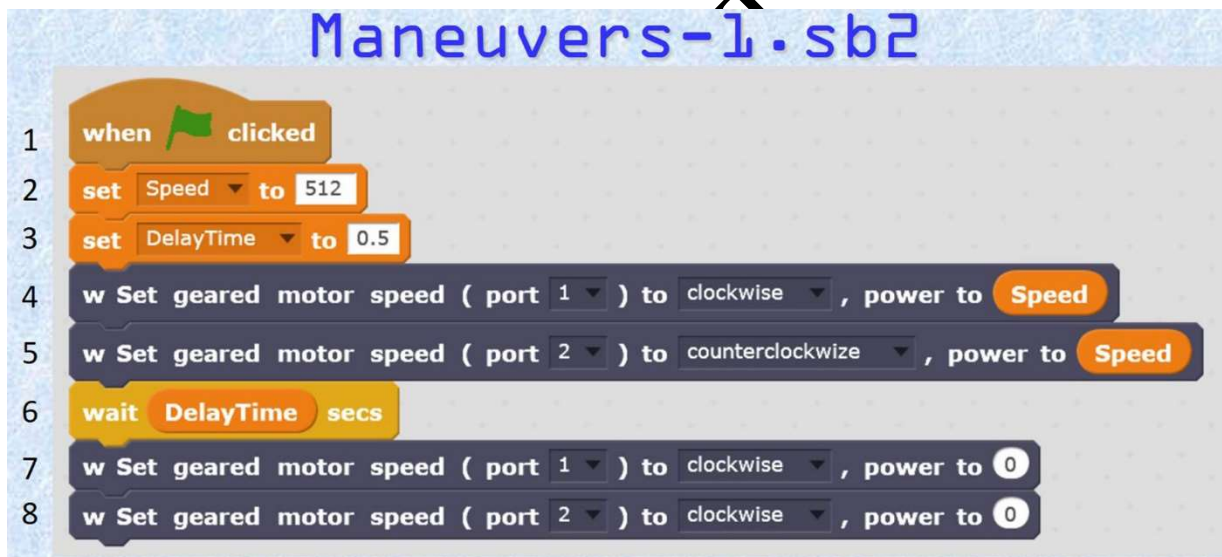


Fig. 4.4 MAIN SCRIPT for “Maneuvers-1.sb2” project.

Copy

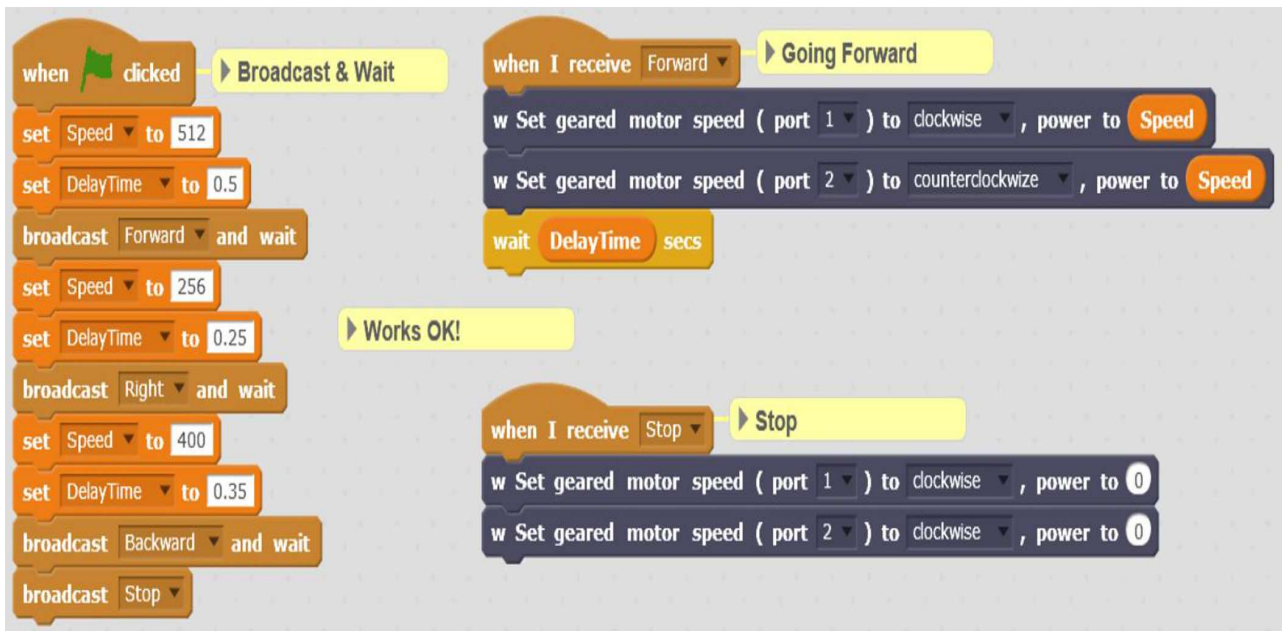


Fig. 4.6 MAIN SCRIPT and two Maneuvers SUB-SCRIPTs in “Maneuvers-4a.sb2” project.

4.3 “Spinning Top” Maneuvers using on Center IR Sensor

In this section, we take another step towards robot autonomous control with SCRATCH. The program “IR-Sensor-Speed-1.sb2” links the real-time value of the Center IR Sensor to the Forward motion of the robot (see Fig. 4.8). Not shown in this Fig. 4.8 are the definitions of the Variables “Speed” and “IR” via the DATA menu in the SCRIPTS tab.

Now we are ready for our first implementation of the Sense-Think-Act paradigm previously discussed in Chapter 1 to the SCRATCH environment. The sensing component will be the IR Center Sensor of the “Spinning Top” robot as before.

The tasks assigned to this robot are as follows (review Section 3.3 also):

At start, the robot plays a musical melody (Melody 2) to let the user know that it is “ready for action”, and its Speed initialized to zero.

Next, the behaviors of the robot depend on the position of an object relative to the IR Center Sensor, thus the condition-action list to be considered is as follows:

If the IR Center Sensor reading is less than 50, the robot stays still.

If the IR Center Sensor reading is between 50 and 300, the robot spins right with its speed proportional to the IR Center Sensor reading and it plays a musical instrument for 0.5 beat.

If the IR Center Sensor reading is between 300 and 500, the robot spins left with its speed proportional to the IR Center Sensor reading and it plays a second and different musical instrument for 0.5 beat.

If the IR Center Sensor reading is larger than 500, the robot rolls forward at Speed=512 to escape from the object and it plays a third musical instrument for 0.5 beat.

This condition-action list indicates that we are dealing with a situation whereas one and only one condition can happen at any one time during the runtime performance of this robot, thus we will need to use an IF-ELSE-IF structure for our solution to this problem.

The next SCRATCH project “Avoider-3.sb2” offers a different solution technique to these maneuvering tasks. **This new approach still relies on the same condition/action listing as described at the beginning of this section 4.4, but it distills different behavior patterns from this list:**

- 1) We must identify a “default” action which in this case is “Going Forward”.
- 2) We will be using several simple IFs in parallel instead of a single big IF-ELSE-IF structure like before. **We also need to do a deeper analysis of the condition/action list to retain only the “essential” condition/action pairs.**
- 3) We also **need to apply a Delay Time to each action made** (this part was not needed when the IF-ELSE-IF structure was used). This Delay Time can be tuned for optimal performance.

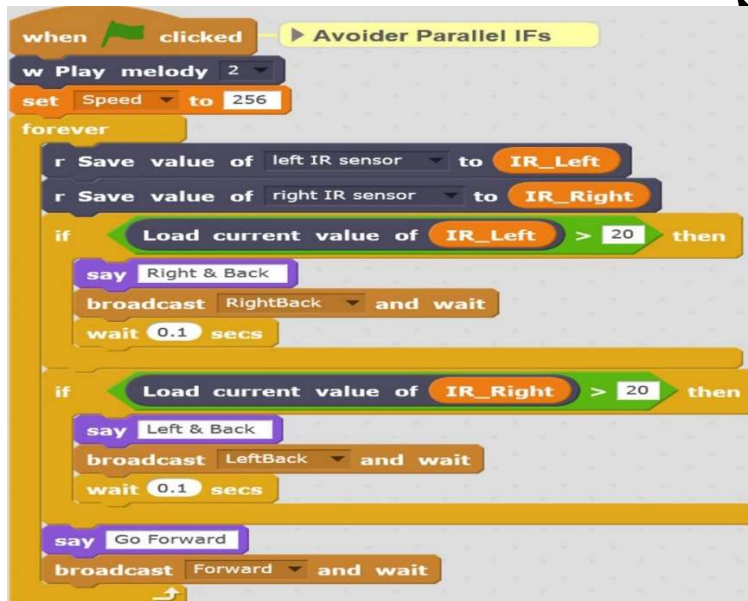


Fig. 4.14 Main SCRIPT used in program “Avoider-3.sb2” (parallel IFs used).

Fig. 4.14 shows the revised FOREVER loop:

At the start of the FOREVER LOOP, Variables IR_Left and IR_Right are updated with the latest data input from the IR Left and Right Sensors. Next are two parallel IF blocks.

The first IF block checks only on the condition for an object on the robot’s Left (conditional expression “IR_Left > 20”). If this is TRUE, a “BROADCAST & WAIT” block is used to make the robot back up and turn right to steer itself away from the obstacle. A short delay of 0.1 second is necessary for the robot to stay in this “action” to make this algorithm work properly. The reader can try different values for the time delay to find the optimal robot performance.

The second IF block represents the condition when there is an object on the robot’s right, then the robot backs up and turns left to steer itself clear from the obstacle, using another “BROADCAST & WAIT” block with the message “LeftBack”. A short delay 0.1 second is also necessary for the robot to stay in this “action” to make this algorithm work properly.

Please note that the “default” maneuver is always programmed as the “Last Action” (“Going Forward” in this case).

4.5 “Follower” Maneuvers using Left and Right IR Sensors

To round out the reader’s experience with using the Left and Right IR Sensors, let’s apply a “reverse logic” on them, essentially let’s make the robot follow an object when it detects something in front of it.

Let’s first learn how to track a moving object - when it is detected. The corresponding condition/action list can be as follows:

If there is no object detected, the robot should stay still (default behavior).

If there is an object detected by the Right IR Sensor, the robot should swing right.

If there is an object detected by the Left IR Sensor, the robot should swing left.

Should we also plan for the case when an object is detected and located right in front of the robot? Should it “stop” or should it show that it already “locks in” on the object in some other way? From the preceding Avoider SCRATCH coding work, we should be able to let the robot bounce between its two actions of swinging left and right to show that it got the object in its sight, right?

4.6 Remote Control of CarBot via PC’s keyboard

The SCRATCH/R+SCRATCH tool chain does not support a Virtual RC-100 Remote Controller like for the TASK tool (see Section 3.6), as SCRATCH already has a block that monitors the PC’s keyboard, called “When xxx Key Pressed” and it is available in the EVENTS menu of the SCRIPTS tab (see Fig. 4.20).

Another big difference between the RC-100’s behavior and the PC’s keyboard’s behavior is in the way multiple buttons or keystrokes are handled. At any one time, the RC-100 sends over to the robot a 10-bit message that represents the status (OFF or ON) of all its 10 buttons (UDLR123456), thus the user needs to create special TASK codes to separate specific data from each button before ones can use them in a remote-control scheme (see Section 3.6). On the other hand, when multiple key strokes are made on the PC keyboard, the SCRATCH IDE receives them as a stream of individual keyboard inputs, i.e. none of the communications processing techniques created for TASK in Section 3.6 is transferrable to SCRATCH.

In SCRATCH, the general approach is to create independent SUB-SCRIPTS to handle each key stroke of interest, and a MAIN SCRIPT to handle the interactions between keystrokes. The project “RC-Direction-Speed-1Button-a.sb2” shows how to use the Up-Down-Left-Right Arrow keys for affecting the moving directions of a typical CarBot (see Fig. 3.55) and the keys “s” and “f” to switch respectively between a Slow Speed and a Fast Speed regime.

4.7 Line Follower – Obstacle Avoider Project

In this section, we’ll develop a SCRATCH solution equivalent to the TASK program “LineFollower-ObstacleAvoider.tskx” from Section 3.7. It is recommended that the reader reviews Section 3.7 to recall the behaviors of the Left and Right IR Sensors and to understand the threshold values used in the program “LineFollower-ObstacleAvoider.sb2”. The same algorithm, described in detail in Section 3.7 (which won’t be repeated here), is used for the SCRATCH solution, of course with the syntax adapted to the equivalent blocks allowed in SCRATCH.

Fig. 4.26 describes the MAIN SCRIPT of “LineFollower-ObstacleAvoider.sb2”:

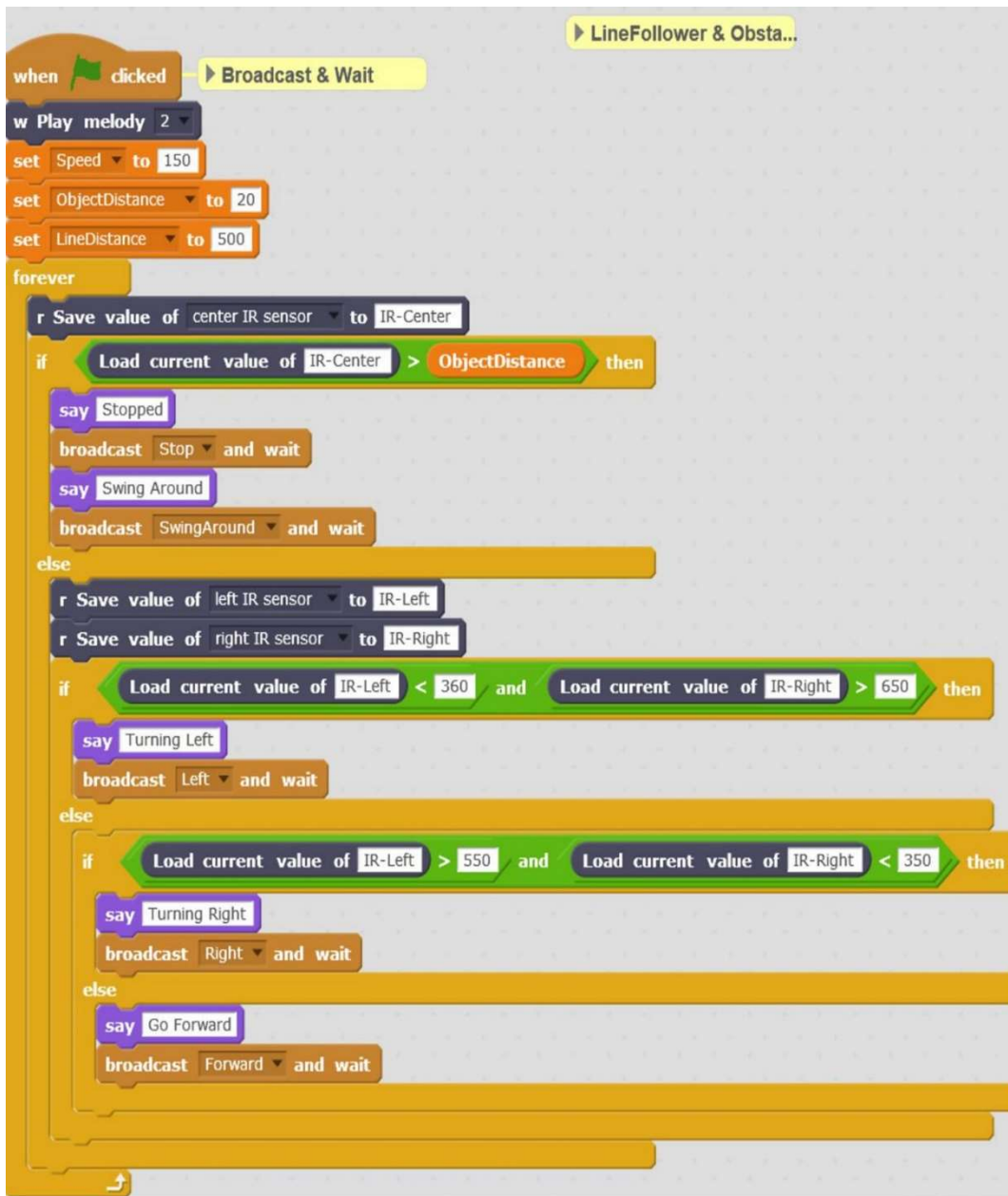


Fig. 4.26 MAIN SCRIPT in "LineFollower-ObstacleAvoider.sb2".

4.8 Moby Dog Multimedia Project with SCRATCH

We will be using a 4-Legged DogBot for this project (see Fig. 4.28). It would be interesting for the reader to compare this SCRATCH version to the TASK/R+m.PLAY700 version in Section 3.9.

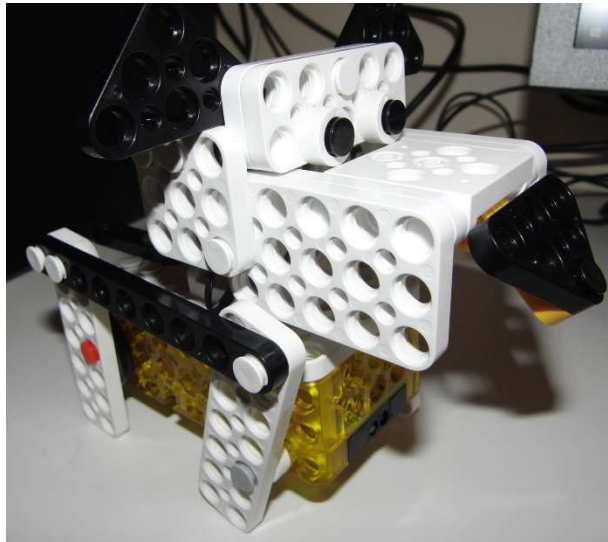


Fig. 4.28 4-Legged DogBot for Project “MoodyDogBot.sb2”.

Fig. 4.30 lists the blocks used in the MAIN SCRIPT:

```

when clicked - Multimedia version
switch costume to 4LeggedDog_s
w Play melody 10
wait 5 secs
forever
  say I am a moody fella~
  w Set sound detected count to 0
  r Save value of sound detected count to claps
  repeat until Load current value of claps > 0
    r Save value of sound detected count to claps
  if Load current value of claps = 1 then
    broadcast friendly and wait
  if Load current value of claps = 2 then
    broadcast angry and wait

```

Fig. 4.30 MAIN SCRIPT in Project “MoodyDogBot.sb2”.

Once the GREEN FLAG is clicked at runtime, the DogBot SPRITE switches to its 4 Legged Costume in the STAGE area, and the CM-50 is commanded to play Melody 10, then SCRATCH IDE waits for 5 seconds for Melody 10 to finish playing.

The FOREVER LOOP is entered and the DogBot announces that “I am a moody fella~”.

Next SCRATCH sends out commands to the CM-50 to set its “Sound Detected Count” to zero, then reads in the current value of “Sound Detected Count” and saves it into Variable “claps”. At this point, “claps” has a zero value, thus when the REPEAT UNTIL loop is entered next, the conditional expression (claps > 0) evaluates to FALSE, therefore this loop is entered. Next is a read command to the CM-50 to refresh the value of Variable “claps” and the conditional expression (claps > 0) is checked again. If the user stays real quiet, this REPEAT UNTIL would keep on looping. Please note that the structure, composed of an **initial** “Save Value Of” block, a “REPEAT UNTIL” **loop** and an **internal** “Save Value Of” block, is a very effective way to monitor a Sensor reading until a certain threshold is reached, causing the REPEAT UNTIL loop to quit. The reader will see this structure again and again.

When the user starts clapping, the REPEAT UNTIL Loop is exited, and the First of the two parallel IFs is processed to check on whether “claps” equals 1? If this is TRUE, a (BROADCAST “friendly” and WAIT) is launched to trigger the SUB-SCRIPT “friendly”.

Copyrights 2020 CNT Robotics LLC

Chapter 5: Selected Mechanical Design Concepts

Although the parts in the PLAY700 kit are made of plastic, they are designed to mimic standard mechanical parts found in everyday life. Thus, there are many sources of information on mechanical engineering and design that the readers can draw from.

5.1 Wheel Based Applications

5.1.1 Passive Front Supports/Wheels

As a typical PLAY700 CarBot has only two drive wheels, at the minimum it would need some Single Bottom Support (see Fig. 5.1). For smooth hard surfaces, these are OK to use, but they would dig into softer surfaces like carpet or sand and dirt.



Fig. 5.1 Two types of Single Bottom Support for CarBot.

The small White Washers can also be used as Side Supports (see Fig. 5.2), as these washers can “freely” roll, they would reduce friction with the ground surface and thus the CarBot can be more maneuverable.

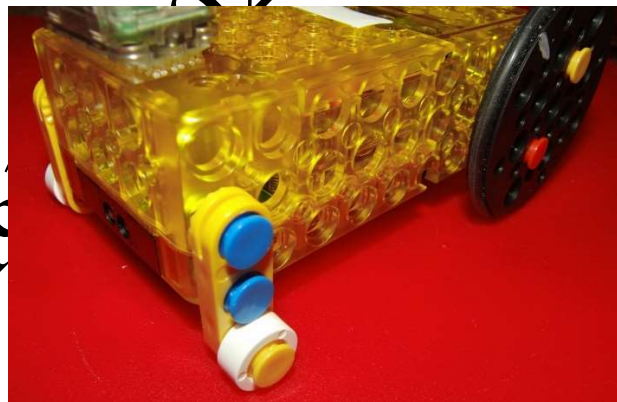


Fig. 5.2 White Washers as Front Wheels.

5.1.2 Four-Wheel Drives

Removing the Black Tires from the Large Driving Wheels and they become Pulley Wheels, and using some Rubber Bands to wrap over these Pulley Wheels, we can get a 4-Wheel Drive CarBot (see Fig. 5.4).

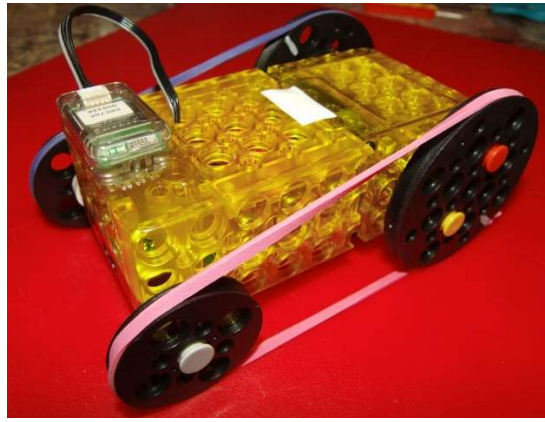


Fig. 5.4 Four-Wheel Drive CarBot using Pulley Wheels and Rubber Bands.

Adding “Spikes” to the Front Wheels would make this CarBot handle carpeted floors better or climb over small obstacles (see Fig. 5.5).

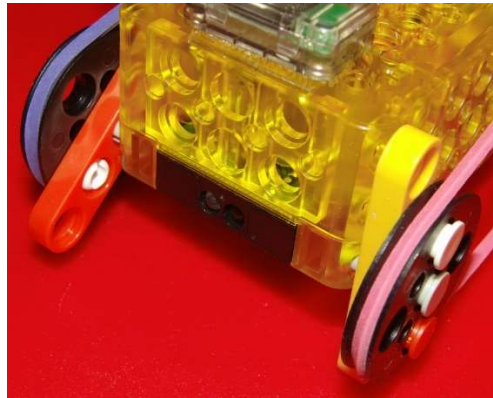


Fig. 5.5 Adding “Spikes” to Front Wheels.

Alternatively, adding “Spikes” to the “Spinning Top” robot would make it into a “Dancing Bear” or a “Waddling Penguin” (see Fig. 5.6).

The PLAY700 kit does not actually have “gear” parts, but we can use the Large and Small Pulley Wheels and the Large Rubber Tires to build a Gear Train as shown in Fig. 5.7.



Fig. 5.7 Gear Train built from Pulley Wheels and Rubber Tires.

This Gear Train can be used to illustrate concepts such as Gear/Speed Ratio, Direction Reversal and Power Transmission. The Rubber Tire plays a very important role by its being “squeezed” at the contact points

between adjacent Pulley Wheels, as this creates "friction" between the wheels which then allows rotational motions on one wheel to be transmitted to the other wheel (but in the reverse direction).

Some readers probably already notice that if an identical gear train to the one shown in Fig. 5.7 is mounted on the other side of the CM-50, then we've got ourselves a 4-wheel drive as shown in Fig. 5.9. This version of 4-wheel drive would be more powerful and maneuverable than the one using rubber bands shown in Fig. 5.4, because the coupling via the pulley/tire interface is mechanically more efficient and the rubber bands can be too "flexible" to transmit high power needs. This "new" solution however requires the user to acquire two more Large Pulley Wheels and two more Large Black Tires beyond the parts that came with the original PLAY700 kit!



Fig. 5.11 Rough Terrain version.

With the Monster Truck and Rough Terrain versions, we can now distinguish as separate the power transmission component (black pulley wheels and rubber tires) from the attached wheels themselves. Alas! These "Gear-based" 4-Wheel Drive, Monster Truck and Rough Terrain versions require more spare parts beyond a single PLAY700 kit!

5.2 Linkage Based Applications

5.2.1 Four-Legged DogBot

The original PLAY700 DogBot is essentially a CarBot (see Fig. 5.12), but it can be modified using a 4-bar Linkage system to become a 4-Legged DogBot as shown in Fig. 5.13. Each side of the DogBot's body uses an independent 4-bar Linkage system.

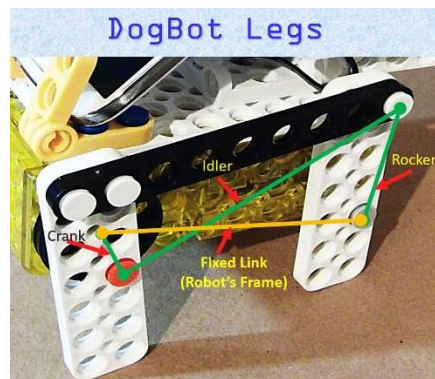


Fig. 5.13 4-Legged DogBot using 4-Bar Linkages.

Wikipedia has a compact write-up about 4-Bar linkages where the reader can learn more about this subject (https://en.wikipedia.org/wiki/Four-bar_linkage). The “Fixed Link” is usually the easiest one to identify as it is “fixed” on the robot frame and 1 of its 2 joint nodes is on the axle of the motor (see Fig. 5.13). This “motor” node is also part of the “Crank Link” which is usually identified with the motor’s horn. The “Idler Link” could be the hardest to find as it does not always map to the obvious physical part that is used for it (see black linkage part in Fig. 5.13 and red linkage part in Fig. 5.14). The “Rocker Link” is usually the “leg” or “arm” for the robot, so it is relatively easy to identify.

5.2.2 Scorpion

The PLAY700 Scorpion robot has three 4-Bar linkages merged together to form 4-legs on each side of this robot (see Fig. 5.14):

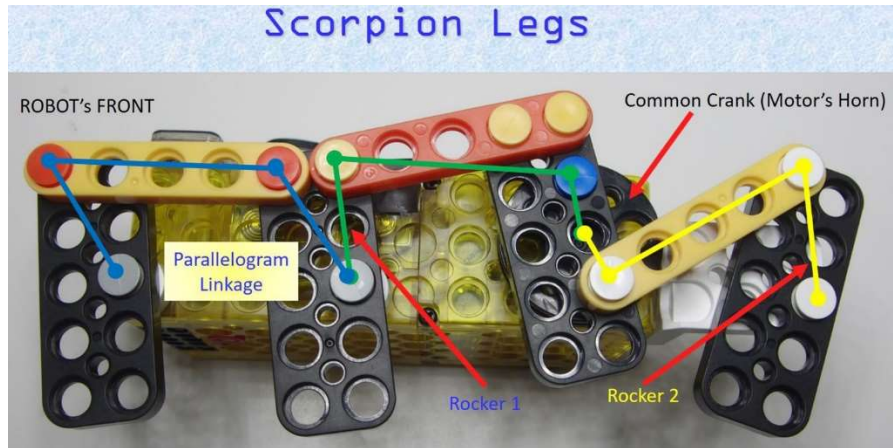


Fig. 5.14 Scorpion’s Legs use three 4-Bar Linkages.

Going from the Robot’s Front, the first 4-Bar linkage is a special one called “Parallelogram Linkage” (for obvious reasons). It is the Blue linkage.

The second 4-Bar linkage is colored Green. It is of the Crank-Rocker type and its Rocker is named Rocker 1 in Fig. 5.14.

The third 4-Bar linkage is also of the Crank-Rocker type, and it is colored Yellow. Its Rocker is labeled Rocker 2 in Fig. 5.14.

Although, in Kinematics sense, there are two separate Cranks - colored Green and Yellow - they are physically attached to the same physical part which is the Motor’s Horn (the only “powered” link for both Crank-Rocker linkages).

5.2.3 Two-Arms Robot

A mixed “Gear/4-Bar Linkage” application is shown in Fig. 5.15 for a Two-Arms robot (nicknamed “Zombie”).



Fig. 5.15 A Two-Arms robot - Zombie.

5.2.4 Monkey's Crawling Motion

Figure 5.16 describes a Monkey robot that can crawl on all four, using its arms and legs. On each side of the Monkey's body, a typical Crank-Rocker mechanism is used. The Monkey's original design can be found in the R+Design Gallery for the PLAY503 kit (Monkey A).

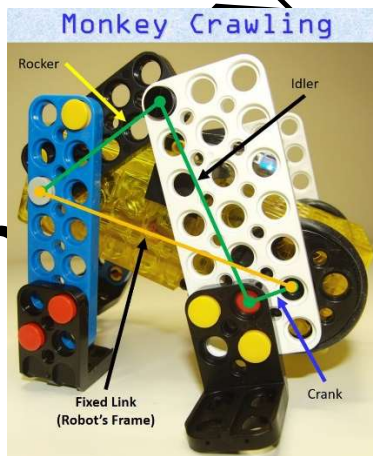


Fig. 5.16 Monkey Robot Crawling Mechanism.

5.2.5 Seal's Hitching Motion

Figure 5.17 describes a Seal robot that uses another "Crank-Rocker" mechanism for its flippers to reproduce the "hitching" maneuver used by real seals to move about on rocks and ice. Similar with the Monkey, each side of the Seal body has its own "Crank-Rocker" system, but this time, the Rocker is a 3-D structure that is "common" to both Left and Right 4-Bar Linkage systems. Because of this physical constraint, this Seal robot can only use its flippers to move forward or backward (i.e. synchronized turning out of the left and right motors), and it cannot turn left or right as these motions would require differential turning of the motors. The Seal's original design comes from the PLAY501 Kit (Seal B).

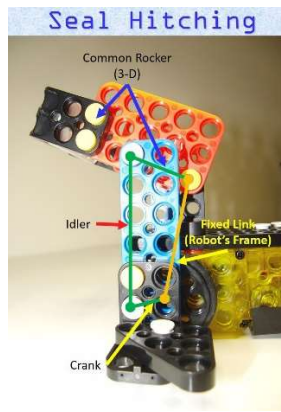


Fig. 5.17 Seal Robot Hitting Mechanism.

5.2.8 Goldfish's Tail Flipping Motion

The Goldfish robot shares the same design approach as for the Seal because it also uses a common Rocker system that links the Left and Right sides of the robot (see Fig. 5.18). Interestingly, this common Rocker has several more components than the one for the Seal.

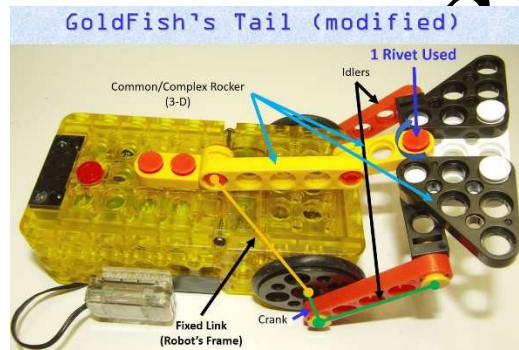


Fig. 5.18 Goldfish Robot Tail-Flipping Mechanism (modified).

5.2.9 Caterpillar's Crawling Motion

The Caterpillar robot is an even more interesting application of the 4-bar linkage concept (see Fig. 5.19). Normally, the Rocker component is connected to the robot's frame via a fixed joint/node as in previous robot designs. However, with the Caterpillar, the Rocker (i.e. yellow 5x9 plate) can "rock" and "slide" on the entire "top back edge" of the CM-50 controller as shown in Fig. 5.19.

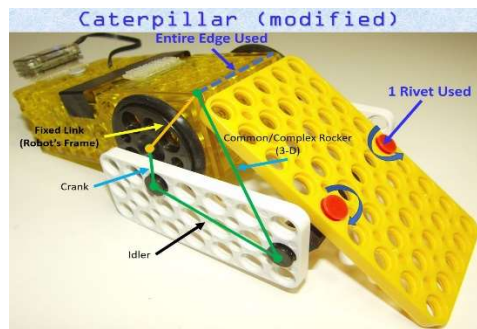


Fig. 5.19 Caterpillar Crawling Mechanism (modified).