Learning Robotics with ROBOTIS DREAM Systems

(Excerpts)

By Chi N. Thai

CNT Robotics LLC, Lawrenceville

2018

Chapter 1: Introduction

1.1 ROBOTIS DREAM II System Overview

ROBOTIS offers quite a large range of educational robotics kits to fit different user's ages and skill sets, please review this YouTube® video (https://youtu.be/Rxmgfose8wE) for a survey of the ROBOTIS "family" of robots.

For the USA market, ROBOTIS offers the PLAY and DREAM robotics systems for students in elementary and middle schools. The PLAY system uses the sturdier 7 mm DIA 1-component rivet system which is better suited to motor skill levels of younger children. The DREAM system uses a smaller 3.5 mm DIA 2-component rivet system which requires more visual/manual skills from the students (see Fig. 1.1). Please note that official ROBOTIS materials prefer to use the "distance between adjacent rivet holes" as the classification parameter, thus "3.5 mm DIA" corresponds to the "6 mm" hole distance, and "7 mm DIA" corresponds to the "12 mm" hole distance.

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

The DREAM II system (http://www.robotis.us/dream/) offers an extensive range of sensors such as Touch, Color, Magnetic, Passive IR and Temperature sensors. It can operate different types of actuators such as continuous-turn and servo motors. It can be made to work with the R+m.PLAY700 App allowing the robot to access multimedia services on mobile devices (see Thai (2018)), or with MIT's SCRATCH 2 to achieve similar goals on PC platforms (this book).

The DREAM II system shares into a common ROBOTIS System Design Paradigm which considers a typical ROBOTIS robot to be a Computer Network with four major components:

1) A Hardware Controller, i.e. CM-150 (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 programmed by the user.

Fig. 1.2 Hardware Controller: CM-150.

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.

Fig. 1.3 Selected Sensors: NIR Sensor (left) and Touch Sensor (right).

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 micro-controllers to work in concert with the Hardware Controller.

Fig. 1.4 Selected Actuators: Servo Motor (left) and LED Display (right)

4) A Remote Device, which can be an RC-100B 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 basic RC-100B is used with the DREAM system, it essentially functions as a NIR-based Sensor (attached to the user) informing the Hardware Controller about which "Up/Down/Left/Right" buttons had been pressed by the user. The basic RC-100B would require line-of-sight access between the physical RC-100B and the NIR receiver (IR-10) attached to the robot. When a smartphone (running the R+m.PLAY700 App) is used with the CM-150, it can act as a BlueTooth Sensor telling the CM-150 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 videos as commanded by the CM-150 according to its programmed logic (see Thai (2018)). When a PC is used as a Remote Device, the issues become more complex. For example, if TASK's PC Virtual Controller (see Fig. 1.6) is used, then the PC acts as a glorified Sensor informing the Hardware Controller CM-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-150 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-150's Firmware (see Fig. 1.8).

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

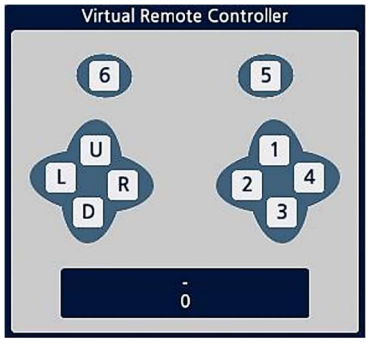

Fig. 1.6 TASK's PC-based Virtual Controller Interface.

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-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-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 running 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.

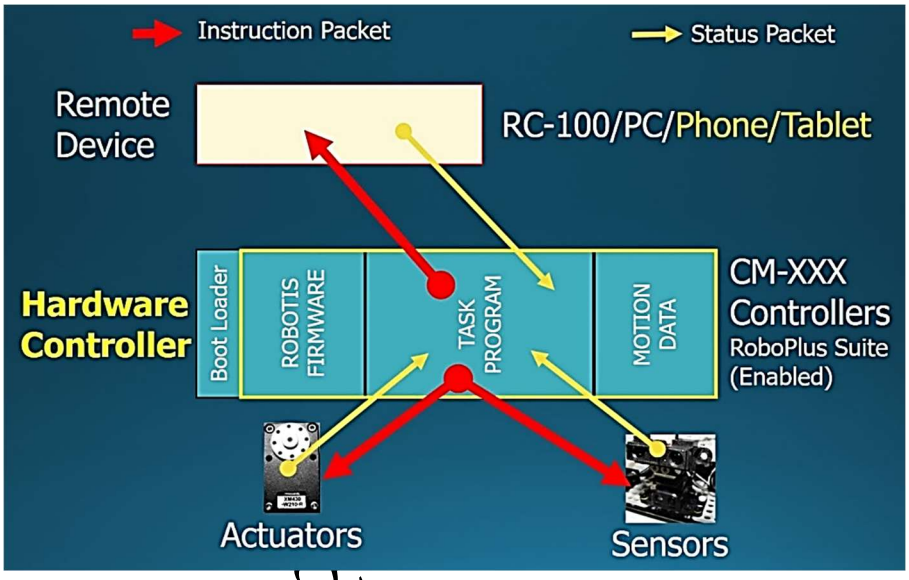

Fig. 1.7 Hardware Controller as Top-Level Controller.

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 ROBOT 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-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).

Chapter 2: Getting Started with DREAM II

Out of its shipping box, the DREAM II School Set is fully functional on a Windows PC platform equipped with a USB port using the provided micro USB cable. This means that the robot will be tethered to the PC via this 3-foot cable during code development, which should be satisfactory for most applications. Once robot programming is finalized, the user can detach the USB cable and use the RC-100B/IR-10 combination for NIR remote control (line of sight required and with a practical range of about 3-6 feet).

If a farther control range is needed (up to 30 feet), the user needs to invest in a set of Master/Slave Blue-Tooth modules which do not require "line-of-sight" and also have reliable "one-to-one" connections within a multiple-robot environment like in a robot competition (http://www.robotis.us/bt-410-set/). The "Master" BT-410 module needs to be attached to the RC-100B remote controller, while the unmarked "Slave" BT-410 module needs to be used on the robot. The "Master" and "Slave" modules are automatically and uniquely paired for the first time when powered within a few inches of each other.

Furthermore, if the user wants to develop and deploy robot codes wirelessly from the PC, then the user needs to also invest in the USB BT-410 Dongle (http://www.robotis.us/bt-410-dongle/), to be connected to an available USB port on the PC. The BT-410 Dongle is a BlueTooth bridge so it can pair to "any" Slave BT-410 module when they are in proximity of each other (1-2 inches). Once paired, they stay in "one-to-one" connection with each other up to 30 feet.

The software packages that the DREAM II kit can use are listed below:

1) R+DESIGN is a 3D CAD tool for users to aid in their assembly of existing ROBOTIS example kits, as well for experienced users to document their own designs using existing ROBOTIS components (http://support.robotis.com/en/software/roboplus2/r+design/r+design.htm).

2) R+MANAGER is used to check on the overall operation of the CM-150 controller and its built-in NIR and Sound Sensors, the external Geared Motors and to some extent the external Servo Motors (more details in Section 2.1). MANAGER can be used to update or recover the firmware on the CM-150 and its peripherals. The MANAGER tool only works on PCs.

3) R+TASK is the software tool used to develop the robot codes and to deploy them onto the CM-150 (see Chapter 3 for a more detailed treatment). A mobile version is available for iOS and Android devices and it is called R+m.TASK.

4) R+SCRATCH is a helper software serving as an interface between MIT's SCRATCH 2 software and the firmware residing on the CM-150 (see Chapter 4 for a more detailed treatment). R+SCRATCH only works on PCs.

This web site (http://www.robotis.us/software/) contains download links for all needed DREAM II software tools.

Next, let's build a Basic Avoider robot as the demonstration platform (see Fig. 2.1) for various software demonstrations in the next sections. Please watch Video 2.1 to see how to use R+DESIGN on the PC to put it together (only up to Step 201 of the DREAM Avoider design).

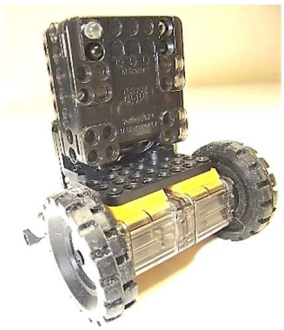

Fig. 2.1 Basic Avoider demonstration platform.

2.1 Using Basic Avoider on Windows PCs

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 the CM-150 Controller and the actuators or sensors attached to it are working properly. Fig. 2.2 shows the main menu for the MANAGER tool when it connects to a CM-150. The column labeled "Control Table" contains various functions that can be tested directly via MANAGER: for example, Sound Detected Count, Ports 1 & 2 Motor Speeds, etc... These functions (and more) can also be programmed via the TASK tool (to be shown later in Chapter 3). Sometimes, the CM-150 may malfunction in some way, and usually the best solution is to "recover or update its firmware" which can only be done via this MANAGER tool. Please see Video 2.2 for more details.

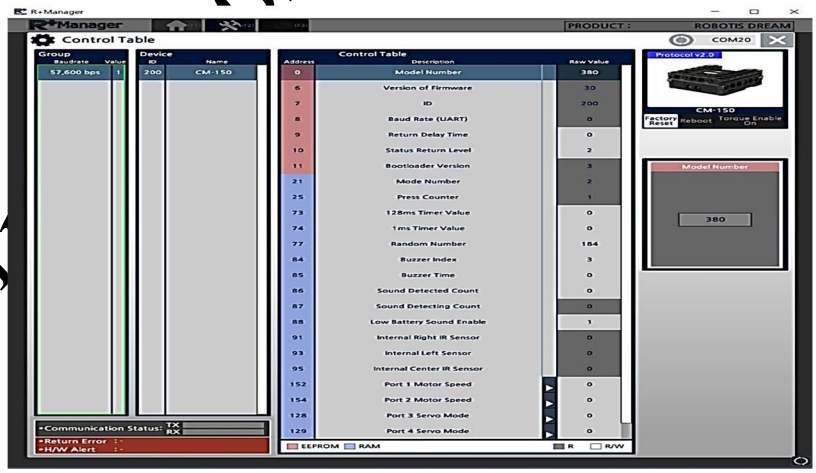

Fig. 2.2 MANAGER Tool on CM-150.

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. 2.3).

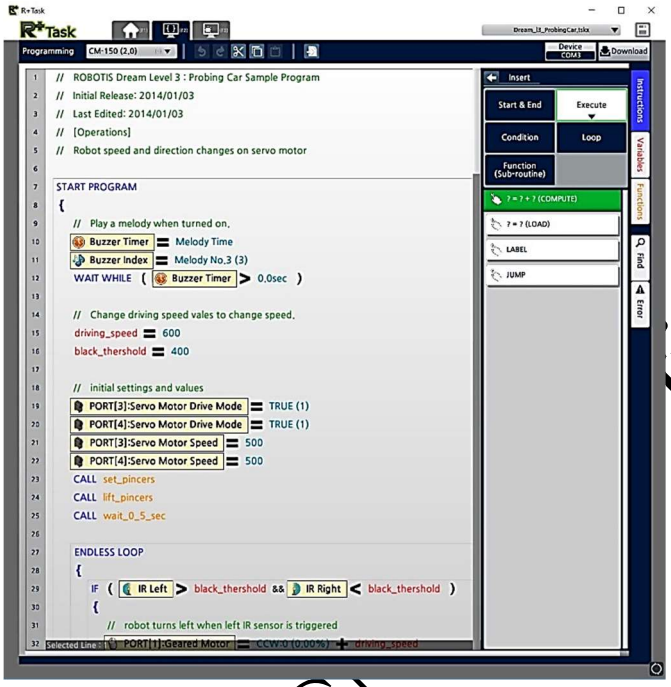

Fig. 2.3 TASK tool on a computer program for CM-150.

Please see Video 2.3 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.

R+SCRATCH (Fig. 2.4) 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-150. Please note that the Firmware on the CM-50 needs to be V.31 or above for R+SCRATCH/SCRATCH2 to work properly with the DREAM II System. Please watch Video 2.4 for a demonstration of its typical use.

Fig. 2.4 R+SCRATCH background application for PC.

SCRATCH 2 is a well-known software tool with millions of 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. 2.5). Please see Video 2.5 for a quick tour of its event-

programming capabilities which are most applicable to robotics. Its proficient use also requires lots of training and practice on the part of the user which will be developed in Chapter 4.

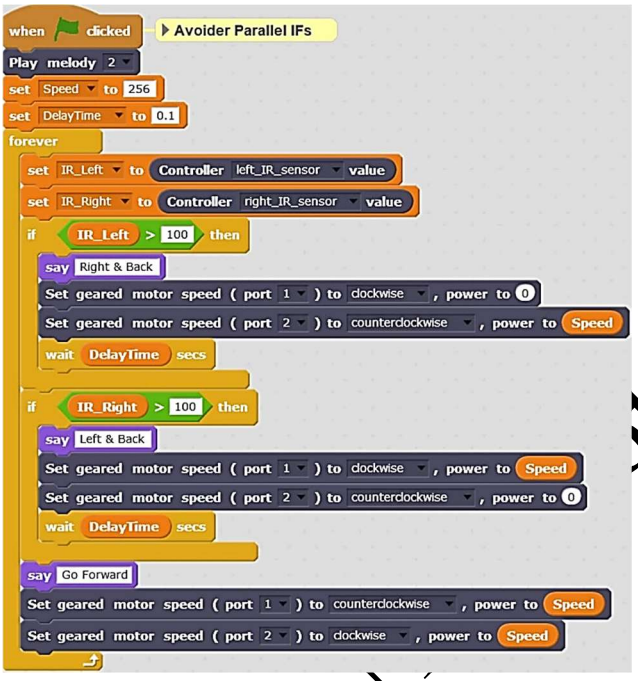

Fig. 2.5 SCRATCH 2 for Basic Avoider project.

2.2 Using Basic Avoider on Mobile Devices

For iOS and Android devices, currently there are 4 ROBOTIS Apps (available in English) that pertain to the DREAM and SMART systems: R+m.DESIGN, R+m.TASK, R+m.PLAY700 and R+m.MOTION (see Fig. 2.6).

Fig. 2.6 Four ROBOTIS Mobile Apps for DREAM and SMART systems.

R+m.MOTION2 is applicable to the SMART, BIOLOID and MINI systems only.

R+m.PLAY700 was originally created to work with the PLAY700 kit only, but the author had found that it can work with the DREAM system as well (see Thai (2018)).

R+m.DESIGN's mobile interface is different from the PC version as previously shown in Video 2.1 (see Fig. 2.7), however its functionality is similar (see Video 2.6).

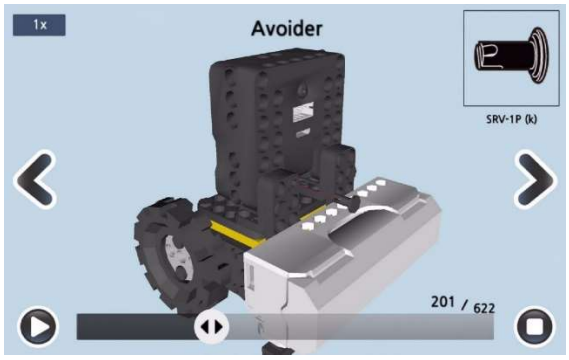

Fig. 2.7 R+m.DESIGN at STEP 201 for Avoider robot.

R+m.TASK's mobile interface is also different from its PC counterpart (see Fig. 2.8), but its overall functionality is the same (see Part 1 of Video 2.7).

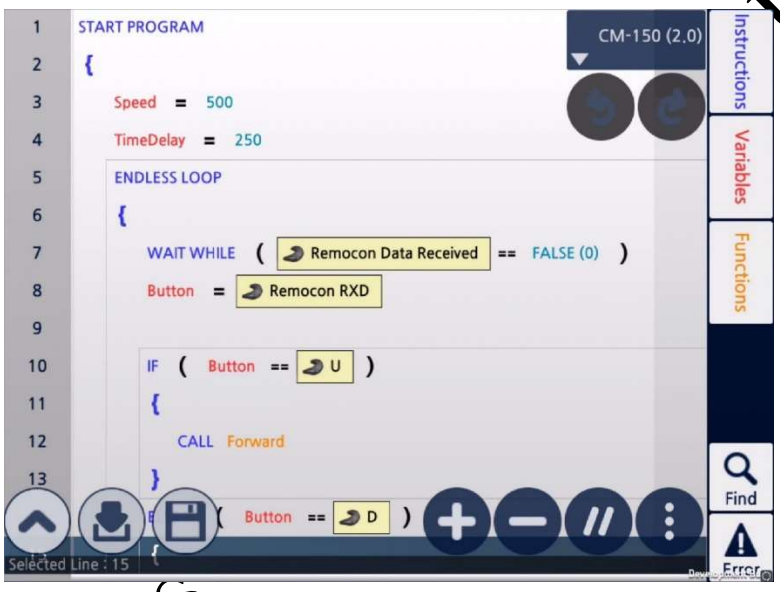

Fig. 2.8 R+m.TASK's Editing Interface.

However, R+m.TASK offers 3 options for its Virtual Controller as compared to only 1 offered on the PC (see Fig. 1.6):

1) A Button-based interface (see Fig. 2.9) where the reader should notice the "greyed out" "in-between" buttons, i.e. between the standard U-D-L-R buttons. For example, the "single" button between "U" and "R" would send the "U+R" value to the robot at run-time. When using the PC-based Virtual Controller, the same effect would require the user to push both "Arrow Up" and "Arrow Right" keys on the keyboard of the PC (see Video 2.3). Furthermore, Buttons 5 and 6 have an "Auto-Pressing" feature, i.e. if the user presses down on them for 2 seconds or longer, they will "lock-in" and stay "pressed down" until the user gives them another quick press to toggle them out of the "Auto-Pressing" mode.

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

This chapter is written for a complete beginner in robotics and computer programming, and we will be using the "Basic Avoider" platform and its variance (Fig. 2.1) to learn how to use the TASK tool. A "Spiral" learning approach (Bergin, 2012) will be used for instructional purposes where the author would show how to use the TASK tool with projects having increasing scope and difficulty.

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 operation)", and the 2 bit-level operators such as "&" (AND) and "|" (OR). Only INTEGER arithmetic is supported. VARIABLE parameters are supported but arrays are not.

2) LABEL and JUMP statements are supported.

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

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

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

Please review Video2.3 or Video 2.7 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-150 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 2.3). Fig. 3.1 is a screen capture of this program which has 5 lines of "relevant" code (lines 4-8). More formally, they are called "Statements". All TASK program begins with the "START PROGRAM" statement (line 2) and an opening curly bracket "{" statement (line 3), and ends with a closing curly bracket "}" statement (line 9).

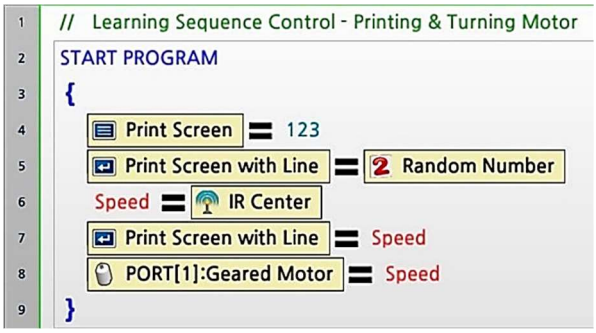

Fig. 3.1 Program "SequenceControl-1.tskx".

All statements between the two matching curly brackets "{}" are considered "legitimate" instructions (but may not be "logically correct") for the CM-150 controller to execute in sequence, top to bottom, at run time. But if they are preceded by the double forward slashes, "//", then they are considered as "Comments", i.e. information for the human reader only and not actual instructions for the controller. For example, Line 1 just tells the reader about the goals of this program. Later, the reader will be shown that it is sometimes necessary to "nest" the curly brackets to create "compound statements" to accomplish certain logical flows needed in a wanted algorithm.

Statements 4 to 8 are called LOAD statements which have this form (i.e. syntax): "A = B". The "=" sign is not the regular "equality" sign that the reader may have seen in algebra. In the world of computer programming, the "=" symbol stands for an "Assignment" operator. Each of the letters/symbols "A" and "B" is called "Operand". Thus "A=B" should be read as ("A" is assigned with the Value of "B"), or it can also be read as ("Value of B" is assigned to "A"). Essentially, "B" should be a Data Value from a Numerical Constant (e.g. "123" in Statement 4), or a Source of Data Input (e.g. a Random Number generated by the Controller internally in Statement 5, or a Value read from a Sensor such as IR Center in Statement 6).

Statement 6 also illustrates the application of a user-defined Named Variable called "Speed" which was first used as an Operand A. Afterwards, "Speed", to be exact its numerical value, can be used as a type of Operand B in Statements 7 and 8.

In other words, the beginner programmer should get used to "read" a LOAD statement from the Right-Hand-Side (RHS) to the Left-Hand-Side (LHS) (which may be "unusual" to English readers). As a side note, TASK supports a second Assignment type of statement called COMPUTE (see its use in a later paragraph) and practically 90% of all the statements in a typical TASK program are of the "Assignment" type.

When the controller executes Line 4 at run time, it would just "assign" its RHS (e.g. the number "123") to the LHS which is the Screen Printer. The user can then see "123" printed out on the Output Screen of the PC or mobile device, but the display cursor will stay on the same output line afterwards, because the wanted command is a "Print Screen".

Next, when the controller executes Line 5 at run time, it will want to execute the RHS first, i.e. it will go to a specific memory location (Address 77 to be exact – and you may remember this from the previous Video 2.2) and it will pull out whatever random number between 0 and 255 that it can find there. Next, it will "assign" this found value to the LHS which is the Screen Printer. The user can then see this random number printed out on the Output Screen of the PC or mobile device (after the previous "123"), but then it will execute a "carriage return" to go to a new line afterwards, because the wanted command is now "Print Screen with (new) Line".

Next, Statement 6 is another Assignment statement where the current value of the IR Center Sensor (Address 95) is assigned to a Variable named "Speed".

Next, Statements 7 and 8 both use the current Value of Variable "Speed" either to print it out (Statement 7) or to assign it as a speed value to the Port 1 Geared Motor (essentially to turn it on if "Speed" happens to be non-zero).

Please see Video 3.2 for more details on how this TASK program "SequenceControl-1.tskx" would actually behave at run time on a Windows PC using the micro USB cable and then with the combination BT-410 Dongle and Receiver. Video 3.2 also shows the run-time performance of this program with a Tablet whereas the author uses the same BT-410 Receiver on the robot and the built-in BlueTooth Master from the mobile device.

Video 3.2 shows the user that because of all the time delays needed for Bluetooth connections to work, the run time behavior of this TASK program was far from satisfactory off the Tablet. On the PC runs, we saw the display of "123" and the random number, but we did not see the Port 1 Motor turn itself on during run time when "Speed" had a non-zero value. On mobile devices, we could not see any "Print Screen" output at all sometimes. The "programming logic" used was correct as the controller could compile and execute its machine code perfectly fine on either programming platform, so what was going on with our robot or setup? **The key understanding here is that everything in real life, and therefore in robotics also, "takes time" to get done, so we'll have to figure out a way for the controller not to "rush through" its program, so that we, slow-poke humans, can "see" what is going on.** So, one way to solve these problems is to make the robot repeats this sequence of actions "endlessly" as suggested by the Sense-Think-Act paradigm (see Fig. 1.12).

The "SequenceControl-2.tskx" program just puts the actions sequence of "SequenceControl-1.tskx" inside an "Endless Loop" and with an additional feature of an incremental Counter which is started at "0" outside the Endless Loop and incremented by "1" every time a loop gets done. Fig. 3.2 shows this completed program and Video 3.3 shows how it can be created by modifying the previous program "SequenceControl-1.tskx".

In "SequenceControl-2.tskx", besides the new "Endless Loop" structure used (Statements 6, 7 and 15, and a nested set of "{}"), another new computing concept of an incremental counter is also illustrated:

a) First, Variable "Counter" needs to be initiated to zero, before entering the Endless Loop. This action can be considered as part an "Init Event", previously mentioned in Section 1.2 and Fig. 1.16.

b) Next, Variable "Counter" needs to be updated for each loop executed, by using a COMPUTE statement to make this Variable "self-add" "1" each time the Endless Loop is executed at run time (see Statement 13). The COMPUTE statement is of the form "A = B *Op* C" which has the standard 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 line 13).

" - " 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.

c) Statement 14 is used to appraise the human user on the current value of the Variable "Counter".

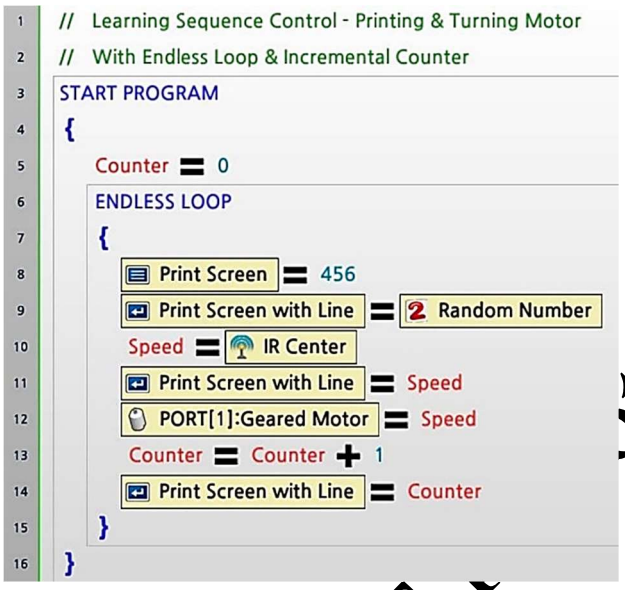

Fig. 3.2 Program "SequenceControl-2.tskx".

Video 3.4 showed that thanks to the Endless Loop, the "Speed" command to the Port 1 Motor (Statement 12) was repeatedly applied to this motor so that we could now see the resulting physical effects of its turning. Essentially, Statement 8 in Program "SequenceControl-1.tskx" (Fig. 3.1) **did** turn on Motor 1, but only for a few microseconds and then the whole robot program terminated, so we could not observe this phenomenon as it was too fast for us to see, actually too short to overcome the friction inside the motor and the mechanical inertia of the robot.

However, ones could see that the visual feedback of the printed output was still too fast for us humans to handle, thus we need to introduce some time delays in the whole process with the use of the Hi-Resolution Timer feature of the CM-150 in the next program "SequenceControl-3.tskx" (see Fig. 3.3 and Video 3.5). The Hi-Res Timer is set at Line 12 for 1.0 sec (or 1000 ms). It functions similarly to a stop-watch or a timer on a microwave oven. To use the Hi-Res Timer, we need first to load the Hi-Res Timer Counter with the wanted time delay (1 second in line 12 – remember Address 74 from Video 2.2?). The Hi-Res Timer Counter would then automatically start to count down to zero with a 1 ms decrement. Next, we need to use a WAIT WHILE statement such as in line 13 to create user-defined time slots associated with certain events. The WAIT WHILE statement is "unusual" in the way that it keeps on executing itself, i.e. "does nothing", as long as the CONDITION specified within the two parentheses "()" remains TRUE, for example (Hi-Res Timer Counter > 0.000 sec) in Line 13. Let's now do a "thought experiment" to see how Lines 12 and 13 can work together at run-time:

When Line 12 is executed, the Hi-Res Timer counter is loaded with 1 second = 1000 ms. Next Line 13 is executed, i.e. the CM-150 is commanded to go to the Hi-Res Timer Counter and retrieve its current value which would be a few milliseconds less than 1000 (let's say 990), as some actual milliseconds have elapsed since the execution of line 12. Thus, the condition (Hi-Res Timer Counter > 0.000 sec) now becomes (990 > 0) and would evaluate to TRUE, therefore the CM-150 would be required to execute line 13 once more. So, this cycle of actions would continue for a while, but you can see that eventually this counter would be at 0. At this point, the condition (Hi-Res Timer Counter > 0) evaluates to FALSE (as

"0 is not greater than 0") and the CM-150 can now quit Line 13 and then redo another loop restarting at Line 7.

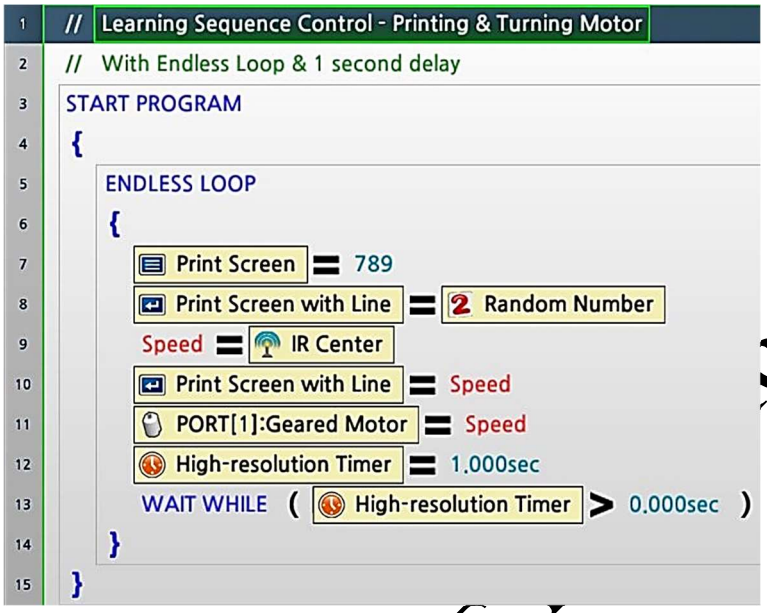

Fig. 3.3 Program "SequenceControl-3.tskx".

Video 3.5 shows how this program performed at run-time on a PC and a Tablet. This video also showed that the IR Center Sensor maximum values were mostly in the 500's range and that the motor was turning at a rather slow speed. This was because the typical Geared Motor can be set to a maximum value of 1023 (i.e. 100% power) and here it was set with values in the 500's (i.e. at 50% power). Thus the program "SequenceControl-4.tskx" was created with a small change in Statement 11 whereas the value of Speed was multiplied by a factor of 4 before assigning this result as the actual speed value for Port 1 Motor – i.e. using a COMPUTE statement instead of a LOAD statement (see Fig. 3.4). See Video 3.6 for the run-time performance of program "SequenceControl-4.tskx".

3.8 Line Tracking & Obstacle Avoidance with NIR Sensors

For this Section 3.8, we will be using a CarBot configuration as described in Fig. 3.37. Construction wise, this CarBot derives from the Avoider (Fig. 3.10), but the whole robot gets rotated down 90 degrees so that the Left & Right IR Sensors are now aimed towards the ground and with additional "sprocket" wheels for front end support. 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.

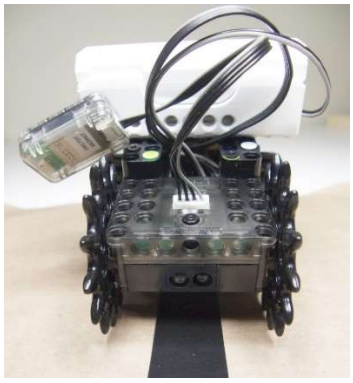

Fig. 3.37 CarBot used as Autonomous Line Tracker & Obstacle Avoider.

Let's first analyze how the three built-in 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. 3.38).

3.9.1 Tricycle Steering Control with Servo Motor

A Servo Motor's operation is very different from the one for the Geared Motor that we had been using. Essentially, a Geared Motor can turn Clockwise or Counterclockwise and at different speeds, while a Servo Motor is used if the user wants to hold a Servo Motor (and the device attached to it) in a programmable but fixed position. Fig. 3.42 looks at the front of a Servo Motor that has an NIR Sensor attached to it. The Servo Motor's position can be controlled by sending it a positive integer between 0 and 1023 whereas the Position Number "512" corresponds the Default "0 degree" Position, while "819" corresponds to +90 degrees and "205" corresponds to -90 degrees. Please watch Video 3.26 for basic use of a Servo Motor.

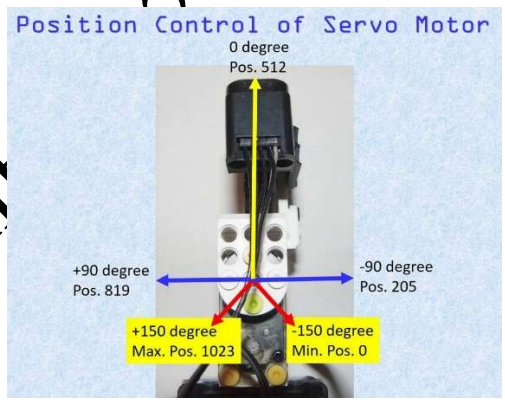

Fig. 3.42 Position Control of Servo Motor with attached NIR Sensor.

The projects in this section use a Servo Motor via Port 3 and an external NIR Sensor via Port 4. The robot platform used is the TriCycle as shown in Fig. 3.43. The TriCycle's powered back end is the same as the previous CarBot (Fig. 3.37) minus the sprocket wheels, but on its front end is mounted a Servo Motor/Front Wheel. The Front Wheel is passive and it is attached via a U-shaped frame to the horn of the Servo Motor. The NIR Sensor is then attached to the Front Wheel assembly (see Fig. 3.44). Thus, the Servo Motor (Port 3)

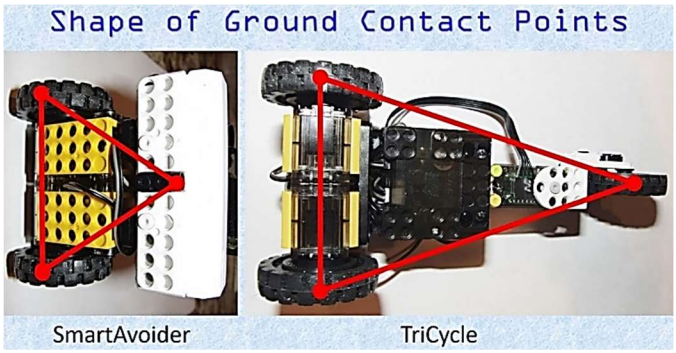

Fig. 3.45 Triangular Shapes of Ground Contact Points for SmartAvoider and TriCycle.

The first TriCycle project is about its Remote Control operations but with a new control approach as implemented in the program "RC-TriCycle.tskx", which itself is based on the previous program "RC-MultiDirectionsSpeeds-Mobile.rskx":

1) The INIT section (see Fig. 3.46) shows how to set up and use the Servo Motor out of Port 3 programming. Line 10 sets the Servo's Drive Mode to TRUE (1), i.e. puts it in Control Position mode *(Note: set this mode to FALSE (0) to use this Servo Motor like a regular Geared Motor)*. Next, Line 11 sets the Servo's Speed to 1023 out of an allowable range of [0-1023]. This Speed controls how fast the Servo can move between programmed Positions which are also in the same [0-1023] range (see their corresponding physical positions as illustrated in Fig 3.42). This Speed value also is a measure of how strong a torque this servo can hold at a programmed position, under the weight/inertia of whatever structure/device that is attached to the same servo. The user could use a mid-value for Speed to achieve less sudden Servo Positions and thus to reduce structure vibrations, but the user needs to use Speed=1023 to hold a steady Servo Position once the servo gets where it is supposed to be. Once the Mode and Speed parameters are set (Lines 10-11), the user can put the Servo Motor to good use by setting its next Control Position value (Line 12) which should be an integer between 0 and 1023 (see Fig. 3.42). There is one important point that beginner users need to be cognizant of when programming servo positional changes: as it will take some actual time periods for the servo to get from one physical location to the next, the user needs to program in some WAIT time for the current operation to finish, before issuing a new command to move the same servo to another location.

2) The INIT section also contains 3 new Variables: SpeedL, SpeedR and SpeedDif (see Fig. 3.46). SpeedL is used exclusively by the Left Geared Motor (Port 1), while SpeedR is reserved for the Right Geared Motor (Port 2). Please see Fig. 3.47 for the use of SpeedL and SpeedR in the new Functions Forward and Backward as examples. SpeedDif will be set proportionally to the Turn Angle of the Steering

3.9.2 Dowel Scanner

For this project, we would need a nimble robot, so it's back to the Avoider's platform, but this time with a Servo Motor "Head" that is slewing a forward-looking NIR Sensor (see Fig. 3.53), yielding a "Dowel Scanner" robot. This robot's tasks are to identify two wooden dowels placed at random in front of it (see Fig. 3.54), specifically to identify which one is closest to it and approach it within a certain distance, then it would need to switch its attention to find the second dowel and likewise to approach it within a given distance.

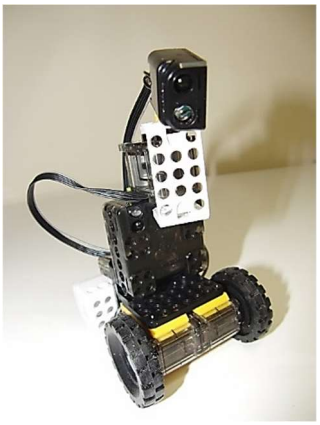

Fig. 3.53 "Dowel Scanner" Robot.

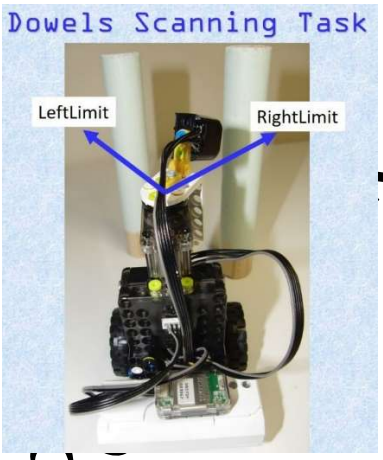

Fig. 3.54 Dowel Scanning Task Layout.

In this project, the NIR Sensor can be slewed in two ways:

1) By the Servo Motor via Port 3 (while the robot is standing still): this technique will be used to identify which dowel (Left or Right) is closest to the robot at the start of the tasks.

2) With the Servo Motor locked into Position 512, thus with the NIR Sensor facing straight forward, the whole robot platform is rotated left-right as needed by the two Geared Motors via Ports 1 and 2 to perform the dowel scanning task using the NIR Sensor, this mode is used in the final identification of each dowel and in the final approach to each dowel.

Figs. 3.53 and 3.54 also show that the NIR Sensor is mounted on its "side" to take full advantage of the fact that the dowels are "vertical" objects. Fig. 3.55 is an output list of the Servo's Position data (first column) and the NIR Sensor's values (second column) at the same positions, as the NIR Sensor is slewed from Left to Right (i.e. Position values from ~700 to ~380), using a simple TASK program.

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), Warner (2015) or Vlieg (2016) for self-learners who want to have a thorough understanding of the SCRATCH 2 language. If the reader is a teacher or trainer 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.

For the CM-150, ROBOTIS designs the software tool R+SCRATCH, **V.1.0.3 or above**, to be a helper application that links the SCRATCH 2 IDE directly to the CM-150 Controller's Firmware, **V.31 or above** (see Figs. 1.8 and 4.1). This means that any TASK code previously downloaded onto the CM-150 is disabled (not erased) as soon as R+SCRATCH connects to the CM-150 (see Part 1 of Video 4.1).

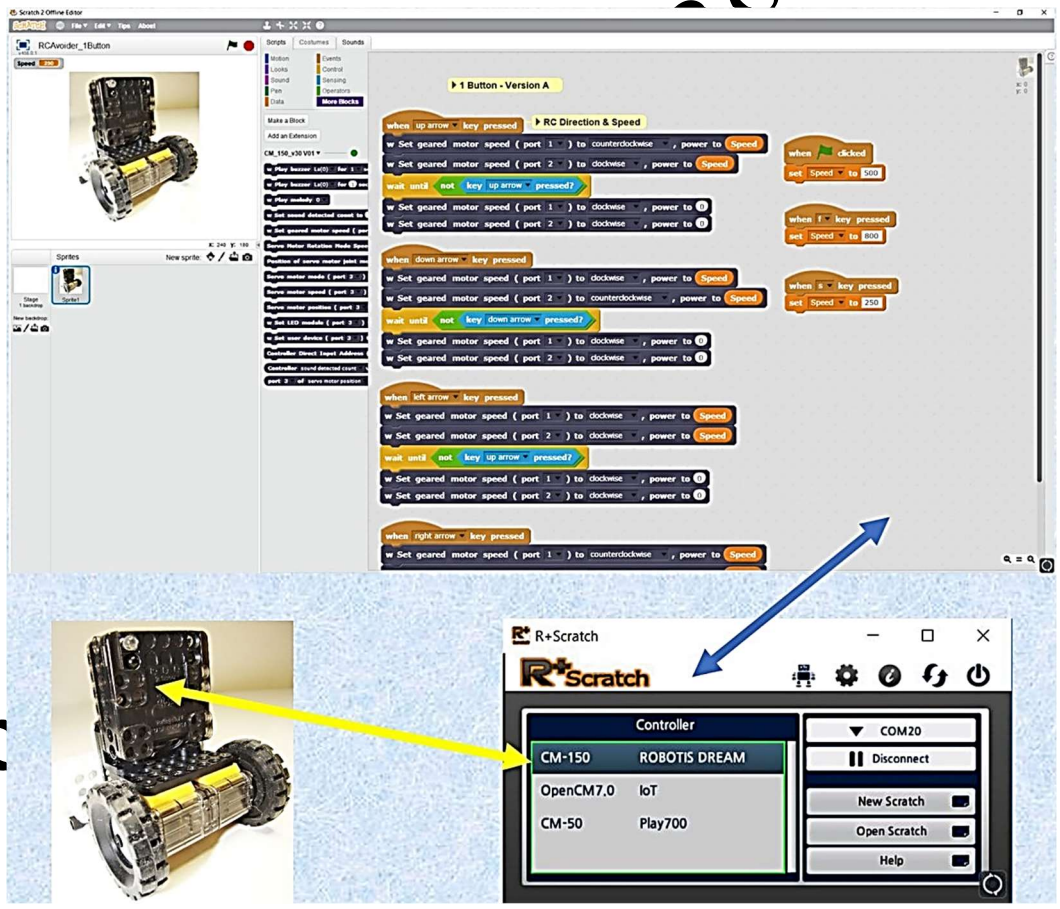

Fig. 4.1 Information Flow between SCRATCH 2, R+SCRATCH and CM-150 Controller.

ROBOTIS also creates a custom SCRATCH HTTP Extension tool (https://wiki.scratch.mit.edu/wiki/Scratch_Extension) to work with the SCRATCH 2 Off-Line Editor to make available 15 SCRATCH Block Codes that can interface with the CM-150's firmware (see Fig. 4.2). The top 13 blocks are of the "WRITE" type of command. Each "WRITE" command requires 50-70 ms delay to progress, from the SCRATCH 2 IDE through the R+SCRATCH helper application, and all the way down to the CM-150 Firmware and actuators mounted on the CM-150 robot. The "Set Value of Address" WRITE block is for advanced users as it will require referencing back to the Control Table of the CM-150 (http://emanual.robotis.com/docs/en/parts/controller/cm-150/). There are only two "READ" type of command, "Controller Value" and "Port Value" which have a pull-down menu to access various devices, internal or mounted onto Ports 3 or 4 (see Fig. 4.3). Each "READ" command requires 120-150 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 and in Section 6.7.

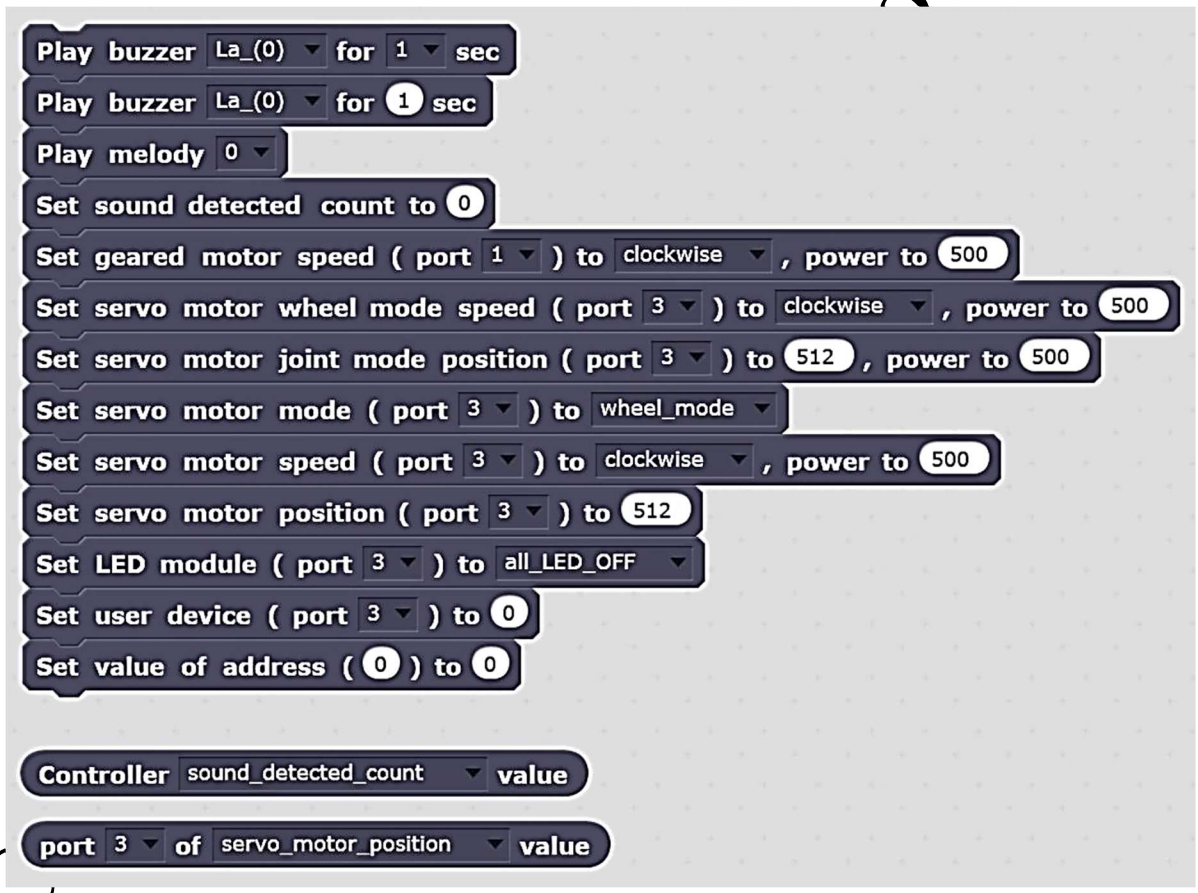

Fig. 4.2 ROBOTIS CM-150 Block Codes compatible with SCRATCH 2 IDE.

4.4 "Spinning Top" Maneuvers using Center IR Sensor

In this section, we take another step towards robot autonomous control with SCRATCH using the "Spinning Top" robot as shown in Fig. 3.10. 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. 4.19). Not shown in this Fig. 4.19 are the definitions of the Variables "Speed" and "IR" via the DATA menu in the SCRIPTS tab.

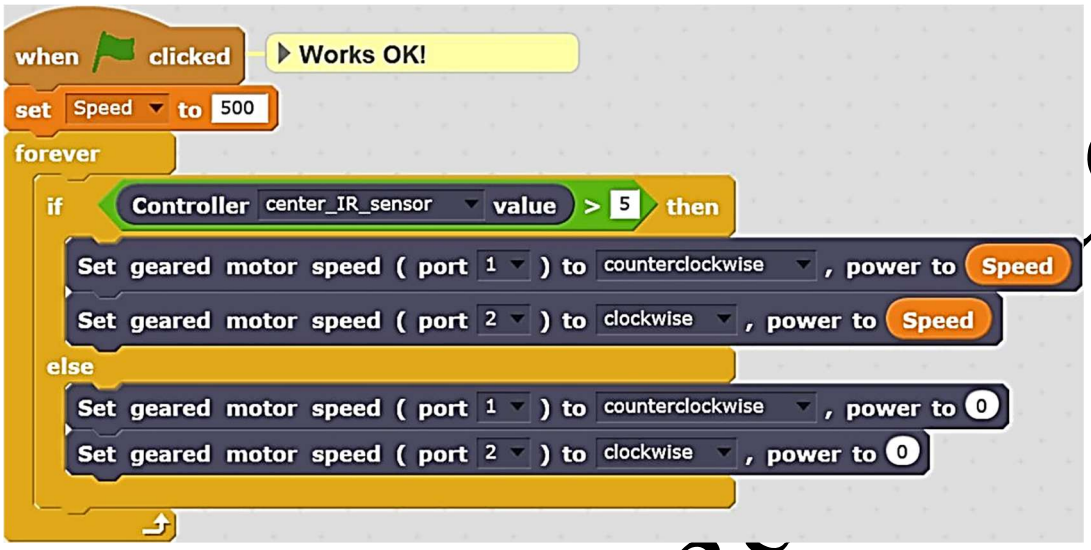

Fig. 4.19 MAIN SCRIPT for "IR-Sensor-Speed-1.sb2" project.

It uses a FOREVER LOOP (same behavior as an ENDLESS LOOP in TASK) which can be found in the menu CONTROL of the SCRIPTS tab:

1) For each iteration of this FOREVER LOOP a READ "Controller Value" block is issued to the CM-150 Controller to return the current reading of its Center IR Sensor which is then used in an IF-ELSE structure (it can also be found in the CONTROL menu). The conditional expression of the IF part is created using the ">" block found in the OPERATORS menu combined with the "Controller Value" block which is accessible from the "MORE BLOCKS" menu. Video 4.9 has the detailed steps for creating such an IF-ELSE structure.

2) During any iteration when this conditional expression evaluates to TRUE, the robot is made to go forward at Speed 500. Otherwise the robot stays still.

The runtime behavior of this program is also shown in Part 1 of Video 4.9: when the robot detects an object close enough its "head", it would dart forward to get away, then it would stop - ready to play again!

4.8 Line Follower – Obstacle Avoider

In this section, we are going back to using the CarBot as described in Fig. 3.37 and we'll develop a SCRATCH solution equivalent to the TASK program "LineFollower-ObstacleAvoider.tskx" from Section 3.8. It is recommended that the reader reviews Section 3.8 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 3.8 (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. 4.35 describes the MAIN SCRIPT of "LineFollower-ObstacleAvoider.sb2":

1) The MAIN SCRIPT starts by setting Variables Speed, ObjectDistance and LineDistance respectively to the Values 350, 50 and 200.

2) The FOREVER Loop is next entered, and the PC reads in the latest "Controller Value" of the Center IR Sensor and use it in the First IF block to check this value against ObjectDistance, to figure out whether there is an obstacle in front of the CarBot or not:

- If there is an obstacle present, the PC stops the bot by turning off power to the 2 Geared Motors. Next it says and initiates the "Swing Around" procedure which essentially rotates the CarBot to the Right for 0.25 second to make sure that the Right IR Sensor is off the black track and on top of the background paper surface (i.e. IR-Right's Value is now greater than LineDistance). At this point in time and in the physical world, we would want the CarBot to keep on rotating right until the Right IR Sensor gets back on top of the black track (i.e when "Right-IR-Sensor" < "LineDistance") and in the other direction of travel. This logical condition is implemented by the First REPEAT UNTIL loop which is designed to be executed if the most recent IR-Right reading is greater than LineDistance, meaning that the CarBot is still seeing the background paper surface and therefore the "Rotate Right" motion should be maintained. This loop effectively does "nothing" because the CarBot is already doing what it is supposed to do, i.e. rotating right (please note the empty slot inside this Repeat Until block).

- Eventually, the CarBot crosses over the black line, and IR-Right's Value would be less than LineDistance, and the First REPEAT UNTIL loop is exited and the Second REPEAT UNTIL loop is taken up. At this point in time, the IR Right Sensor sees the black track, so the Condition (Right_IR_Sensor > LineDistance) is FALSE, so the Second REPEAT UNTIL loop is entered and maintained. Please note that the CarBot is "already doing the right thing", i.e. rotating right, thus there is only an empty slot (i.e. "do nothing") inside the Second REPEAT UNTIL loop. But eventually the CarBot's Right IR Sensor crosses over into the background paper area, then "Right-IR-Sensor" is larger than "LineDistance" and the Second REPEAT UNTIL loop is exited. At that time, the CarBot is physically straddling the black track but oriented in the opposite travel direction than its original one. Next, the PC program flow control pointer is passed to the end of the overall IF-ELSE-IF structure, and the algorithm restarts the FOREVER LOOP once more.

3) If there is no frontal obstacle, the ELSE branch of the First IF would be taken up, whereas a continuing IF-ELSE-IF structure is used to determine three mutually exclusive condition/action pairs:

- IF IR-Left sees Black and IR-Right sees White (i.e. background paper surface), then the CarBot should rotate into the Left to align the CarBot again with the Black Track.

- ELSE, IF IR-Left sees White and IR-Right sees Black, then the CarBot should rotate into the Right to realign the CarBot with the Black Track.

- Otherwise, i.e. the Final ELSE, the CarBot should just drive Forward. At that time, the CarBot could be right on top of the Black Track or it could be right on top of the Background Paper (i.e. it is off-track). This uncertainty is unavoidable due to the placements of the Right and Left IR Sensors with respect to the width of the Black line (see previous discussions in Section 3.8).

Please see Video 4.20 for the runtime performance to this program "LineFollower-ObstacleAvoider.sb2". It is not as responsive as the TASK version (Video 3.25) but much better than the PLAY700 (CM-50) version also in SCRATCH 2 (https://youtu.be/lwpsJu5uRu0).

4.9 Dowel Scanner

In this section, we are going back to using the Dowel Scanner robot described in Fig. 3.53 and we'll develop a SCRATCH solution equivalent to the TASK program "DowelScanner.tskx" from Section 3.9.2. It

is recommended that the reader reviews Section 3.9.2 to recall the behaviors of the Servo Motor (Port 3) and the Frontal IR Sensor (Port 4) and to understand the various Parameter values used in the program "Line-Follower-ObstacleAvoider.tskx". The same algorithm, described in detail in Section 3.9.2 (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. 4.36 describes Part 1 of the INIT Event in "DowelScanner.sb2":

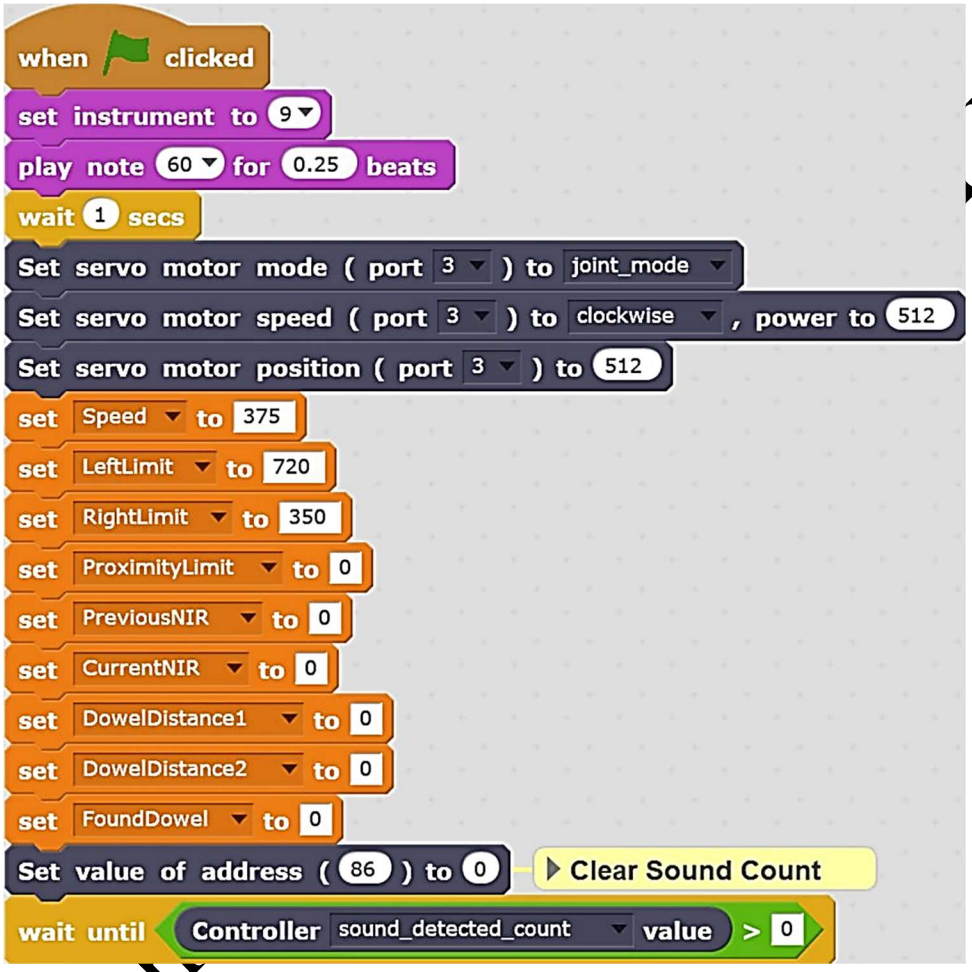

Fig. 4.36 Part 1 of INIT Event in "DowelScanner.sb2".

1) First Note 60 is played on Instrument 9 for 0.25 beat, and the PC is waiting on this action for 1 second.

2) The Servo Motor on Port 3 is set to Joint Mode, its Speed (Power) is set to 512 and it is sent to Position 512.

3) Next a series of Parameters are set to their initial values, from "Speed" to "FoundDowel".

4) The special command "Set Value of Address 86 to 0" essentially initializes the "Sound Detected Count" to zero. The reader needs to consult the file named "CM-150_e_Manual.pdf" (included with the Example Codes ZIP file) and to search for this entry in the Control Table towards the end of this file. The next WAIT UNTIL block just waits for the user to clap his or her hands.

Chapter 5: Mechanical Design Concepts in Level 1 Robots

The parts in the DREAM II kits are designed to mimic standard mechanical parts found in everyday products, e.g. plates, links, rivets, rods, etc… (see Fig. 5.1 for a "Squirrel on Skis"). Thus, there are many sources of information on mechanical engineering and design that the readers can draw from.

ROBOTIS has its own DREAM II materials but some of their web materials are currently available only in Korean (Spring 2018), thus the reader will need to use Chrome as web browser so that it can translate them into "useable" English:

1) ROBOTIS DREAM II materials are at (http://support.robotis.com/ko/product/education/dream2.htm).

2) At present, the R+DESIGN tool has 3D CAD mechanical designs for up to the DREAM I kit only, but the materials for the DREAM II kit will be incorporated into this tool in near future. R+DESIGN can be downloaded at (http://www.robotis.us/roboplus2/) and its user manual can be viewed at (http://emanual.robotis.com/docs/en/software/rplus2/design/). R+DESIGN can be used to design and assemble your own robot design using existing DREAM kits parts.

Cornell University has a comprehensive library of mechanisms at its KMODDL web site (http://kmoddl.library.cornell.edu/). For example, if the reader wants to know more about the 4-bar linkage used for the Ladybug robot, this web page is quite comprehensive (http://kmoddl.library.cornell.edu/).

There are other numerous web sites for mechanism design and simulation, the author is listing just a few below:

http://www.mekanizmalar.com/

https://www.youtube.com/user/thanos010146/videos

https://www.youtube.com/playlist?list=PLhoXNQqrCmEfAaTf0AfQ1Ztxmz2DoZiCk

http://www.roboanalyzer.com/mechanalyzer.html

There should be numerous books about "mechanisms" and "mechanics of machines" if the reader requires more formal references on the web or at your local libraries.

5.1 Direct Motor Drive

In Level 1, the Helicopter is an example of using the Geared Motor directly to move a device such as the 4-bladed rotating wing (see Fig. 5.2).

Fig. 5.2 Direct Drive of Helicopter's Rotating Wing.

5.2 Gear Applications

Technically speaking, the DREAM kit only provides sprocket type of parts (2 sizes – SPS-4 and SPS-17) that can be used to obtain spur-gear functionality (https://en.wikipedia.org/wiki/Gear). Fig. 5.3 shows a typical gear train that can be constructed with these sprocket parts. Although there is quite a bit of "free

play" between the teeth of contacting sprockets, overall there is conservation of kinematic gear ratio: for example, in Fig. 5.3, one turn of the right large sprocket yields one turn of the left large sprocket and in the same direction.

Fig. 5.3 Parallel Gear Train constructed from DREAM kit Sprocket Parts.

Fig. 5.3 is an application where all 3 sprockets' axes of rotation are parallel to each other, but they also can be used in designs where these axes are rotated 90 degrees from each other in a crossed configuration (see Fig. 5.4).

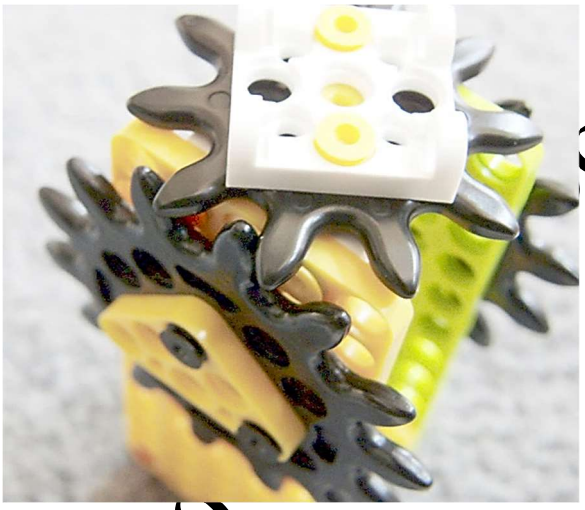

Fig. 5.4 Crossed Gear Train constructed from DREAM kit Sprocket Parts.

The sprockets can also be stacked to provide a more compact gear reduction system as shown in Fig. 5.5.

Fig. 5.5 Stacking Sprockets to achieve a compact gear reduction system.

5.2.1 Using Sprockets (Windmill & Dragonfly)

The Windmill robot combines both parallel and crossed axes designs using 4 sprockets (see Fig. 5.6), while the Dragonfly robot (see Fig. 5.7) uses the same gear reduction system illustrated in Fig 5.5.

Later in Level 3, a Gear Train similar in design to the one shown in Fig. 5.3 is used to obtain a 4-wheel drive CarBot (see Fig. 5.8). More discussions about this Robot are presented in Section 6.9.

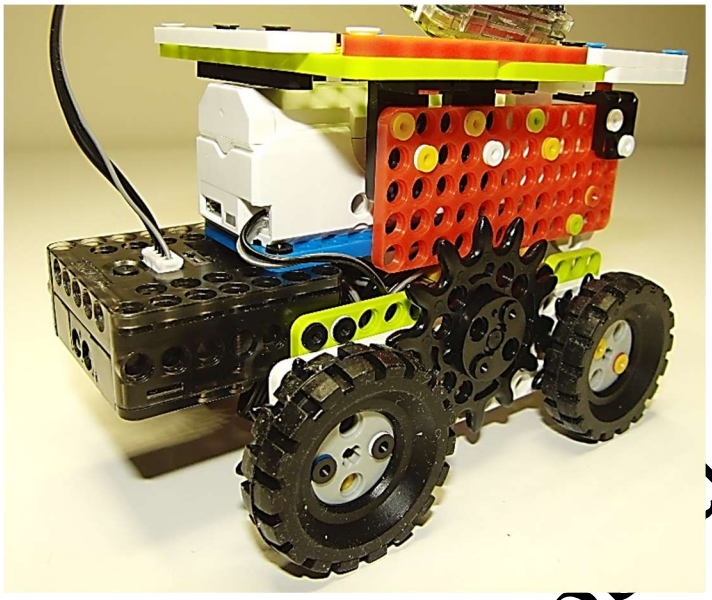

Fig. 5.8 Four-Wheel Drive CarBot from DREAM II Level 3.

5.2.2 Using Pulleys & O-rings (i.e. Tires)

Ones can also obtain "gear functionality" using pulleys and O-rings, i.e. tire systems (see Fig. 5.9), but this system is not as mechanically efficient as the sprocket-based one, as there can be more slippage between the tires contact surfaces. Please note that the big tire on the right comes only from the Bioloid STEM kit.

Fig. 5.9 Gear Functionality using Tire Systems.

5.3 Linkage Applications

To extend the usefulness of Geared and Servo Motors beyond their basic rotational motions, linkage systems can be used (https://en.wikipedia.org/wiki/Linkage_(mechanical)). In the DREAM II system, 4-bar linkage systems and their variants are used to obtain various walking motions. More sophisticated linkage systems have been developed for robot walking motions such as the Klann and Jansen systems (https://en.wikipedia.org/wiki/Klann_linkage; https://en.wikipedia.org/wiki/Jansen%27s_linkage).

5.3.1 Whale

Real whales (and dolphins) swim forward by "flexing" their bodies up and down. This motion is mimicked (as much as possible) with the mechanism shown in Fig. 5.10, whereas the rotational motion of a Geared Motor is transmitted via the Horn/Crank to the Connecting Rod whose tip is constrained to trace an up-down and forward-backward trajectory because of the restricting box frame.

This "swimming" mechanism derives from a slider-crank mechanism (see Fig. 5.11), but without the slider part. The slider-crank mechanism itself is a special case of the 4-bar linkage system (see Fig. 5.12) which is often applied in robot design for the DREAM II series (https://en.wikipedia.org/wiki/Four-bar_linkage).

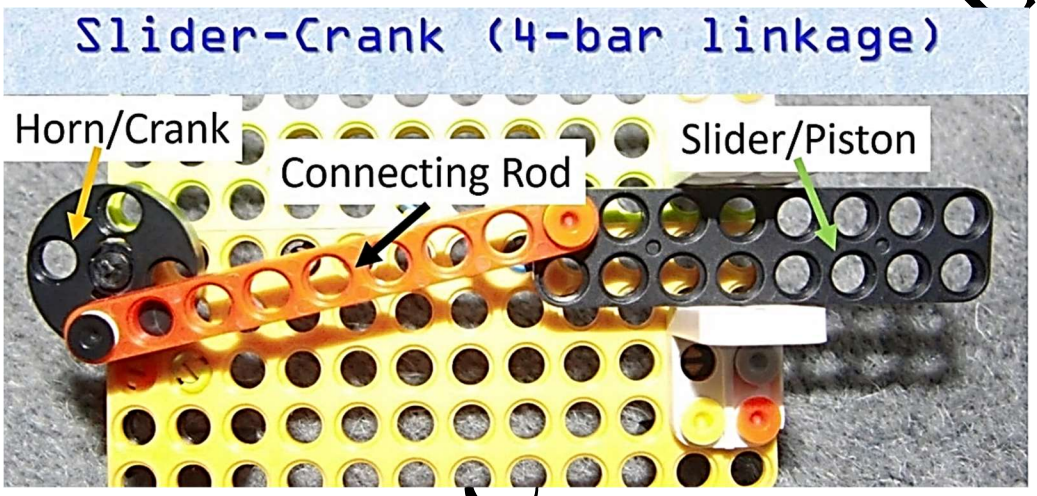

Fig. 5.11 Slider-Crank Mechanism.

5.3.2 Brachiosaurus & Cow

Although physically, the Brachiosaurus and Cow robots look different from each other, kinematics wise both are applications of the 4-bar linkage design (see Fig. 5.13). In this Figure, the author has already identified the various links with different color codes and the reader is encouraged to finish this link identification job by labeling them with their proper names. Please note that both robots walk by shifting their weights in a left-right manner, i.e. their left and right 4-bar systems are in opposite phases. Both robots can go forward or backward equally well. Please note that because the Brachiosaurus is taller in size than the Cow, the battery, its heaviest component, is hung low to help with stability for the Brachiosaurus.

5.3.3 Rabbit

The Rabbit robot also applies the 4-bar linkage design for its 4 legs, but its left and right 4-bar linkage systems are synchronized (i.e. they are in phase). Additionally, the Rabbit has an inclined body position and its Center of Gravity is also positioned just in front of its rear legs, both features contribute to its ability to "hop" forward very well.

5.3.4 Ladybug

The LadyBug has 3 legs on each side of its body, and as a 4-bar linkage system can only provide for 2 walking legs, robot designers would have to merge two linkage systems together. Fig. 5.15 shows that the LadyBug has a 4-bar linkage system for its rear and middle legs, and a 5-bar linkage system for its front leg

and middle leg (which is shared with the rear 4-bar linkage system). The crank link (red) is also shared between the rear and front linkage systems. Same as for the Brachiosaurus and Cow, the Right and Left walking mechanisms for the LadyBug are in opposite phase of each other.

or further illustrations, Fig. 5.16 shows three merged 4-bar linkage systems designed for a Scorpion robot (4 legs on each side) from the PLAY700 kit. For its two front legs, this Scorpio uses a Parallelogram Linkage whereas the Crank and Rocker links have equal lengths, likewise for the Coupler and Fixed links.

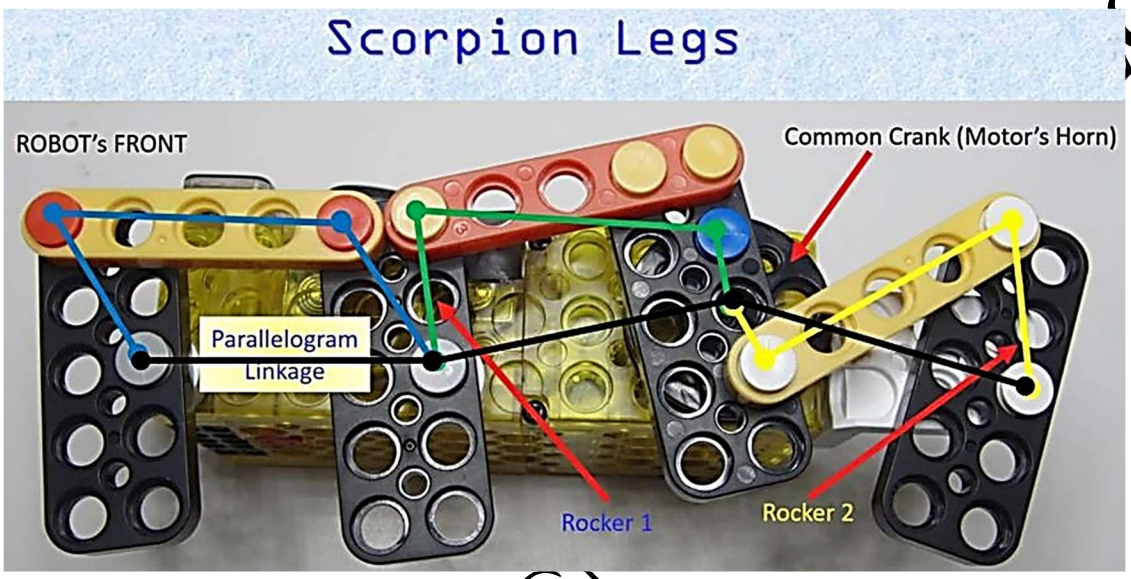

Fig. 5.16 The PLAY700 Scorpion linkage systems.

5.3.5 Chick

For the robots described so far, the walking mechanisms are independent for each side of the robot. The Chick robot design showcases a new approach where the left and right walking mechanisms are linked to each other via an elaborate median mechanical structure (e.g. its head – see Fig. 5.17).

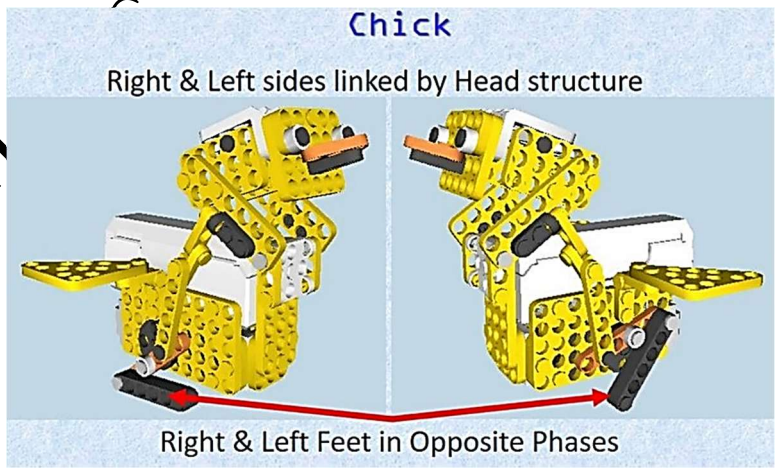

Fig. 5.17 Right and Left Views of the Chick robot.

5.3.6 Kangaroo

Compared to the Chick robot, the Kangaroo robot has a simpler walking mechanism: it also uses a Slider-Crank design for its feet, but the Left and Right sides are synchronized (as the real Kangaroo can only hop around) and it uses its big tail to stabilize itself in the forward-backward direction (see Fig. 5.21).

Fig. 5.21 Kangaroo Hopping Mechanism and Stabilizing Tail Structure.

5.3.7 Caterpillar

The Caterpillar robot mimics the motion of an Inch-Worm which alternates between body retraction and extension moves to achieve a forward (or backward) motion. The Caterpillar is designed with two distinct sections - front and rear, connected by two sets of linkages - top and bottom (see Fig. 5.22).

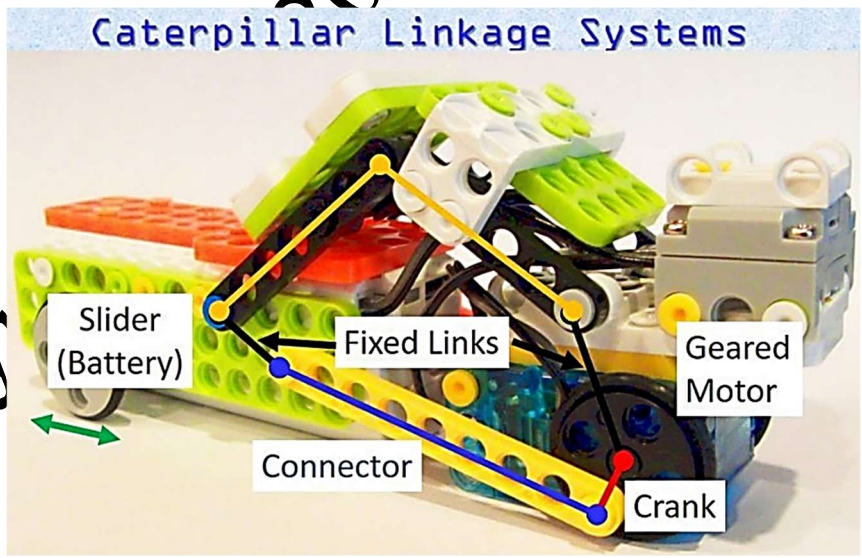

Fig. 5.22 Caterpillar Linkage Systems.

The Front Section has the Geared Motor and the Power Switch module, while the Rear Section has the LiPo Battery and Battery Case. Each Front or Rear Section has its own Fixed links (black links in Fig.

5.22). The Top Linkage system (2 green links) is purely passive and it is used to keep the Front Section down on the ground surface. The Bottom Linkage system is powered via the Geared Motor and it is of the Slider-Crank type. The Crank is the Motor's Horn as usual, and it uses the Connector link to transfer motion to the Slider in the rear. The Slider role is fulfilled by the Battery and Case combination which is also equipped with a pair of passive wheels to minimize contact friction with the ground surface which itself acts as a kinematic constraint.

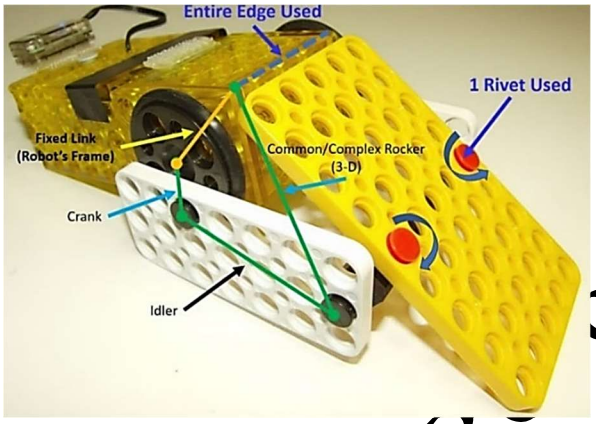

Fig. 5.25 Caterpillar robot from PLAY700 kit.

Fig. 5.25 shows another Caterpillar robot design using the PLAY700 kit. This one can go forward and backward like the DREAM version, but it can also turn left and right (please watch the YouTube video at https://youtu.be/tEK3LcaLrrI to see how it works).

Normally, the Rocker component is connected to the robot's frame via a fixed joint/node as in previous robot designs shown in this Chapter. However, with the PLAY700 Caterpillar, a structurally simpler, but more complex kinematics-wise, Rocker (i.e. yellow 5x9 plate) can "rock" and "slide" on the entire "top back edge" of the PLAY700 controller (see top of Fig. 5.25), i.e. to offer more "degrees of freedom" than the Chick's Head in Section 5.3.5.

Chapter 6: A Closer Look at Select DREAM II robots

The goal of this Chapter is to get the reader beyond the ROBOTIS-provided R+DESIGN example constructions and example TASK codes, by providing additional hardware designs as well as code enhancements whenever possible and appropriate. Most of these enhanced robots will require the user to obtain additional actuators and sensors beyond the ones included in the DREAM II School Set.

6.1 Elephant with NIR Sensor

In all the DREAM robots described so far, the reader should have noticed that the two Geared Motors (GM) were always mounted on the same axis of rotation and robot directional changes were made by modifying the rotation direction of one GM versus the other, and/or by varying the rotational speed of one GM with respect to the other. The Elephant design offers a rather unusual locomotion system, whereas the two Geared Motors axes of rotation are set up perpendicular to each other instead (see Fig. 6.1). One GM (Port 2) drives two small pulley-wheels (Part No. SPO-4X and SPO-5) to move forward and backward, while the other GM (Port 1) provides left and right directional changes using a Medium Tire (Part No. STR-39).

6.2 Flower & Butterfly with NIR Sensor

An NIR Sensor (Port 4) is added to the original robot design called "Flower & Firefly" to result in the "Flower & Butterfly" robot as shown in Fig. 6.7 (bottom right above the white LiPo battery). The author wanted to use a light source to simulate "daytime" (when this robot would operate) and "night-time" (when this robot would "sleep"). Although the TASK tool shows that an Illuminance Sensor is available in its "Set Device" menu (see Fig. 6.8), this sensor is not yet commercially available in the U.S.A., thus the NIR Sensor was used as a substitute sensor for this purpose. The TASK code changes are quite minimal as shown in Fig. 6.9 (essentially an IF-ELSE structure can do the job adequately – Lines 15 and 22).

6.3 Seal with Servo Head

In the original Seal Robot design, the head structure is mounted rigidly onto the robot's body. In this revised project, a Servo Motor (via Port 3) is mounted between the Seal body and its Head (see Fig. 6.10) to allow the robot to move its head from side to side. Please see Fig. 6.11 for details of the construction of the mount for the Servo Motor, and in the previous steps the reader also needed to hook up the 5-pin cable to Port 3 before the assembly of the battery cradle.

Fig. 6.10 Seal Robot modified with Servo Head.

6.4 Beetle with NIR Sensor

Back in Section 3.9.2, we had used an NIR Sensor attached to a servo motor which itself was mounted onto a wheel-based robot, assuring a stable platform, while the sensor was scanning for dowels (see Fig. 3.53). In this Section 6.4, we are going to explore the performance of a similar NIR dowel scanning system which has the NIR Sensor mounted onto a "walking" Beetle robot instead (see Fig. 6.13). Because of the 4-bar walking mechanisms used on its Left and Right sides, the Beetle rolls from side to side when it is in motion. This rolling motion in turn affects the NIR Sensor's view port and thus may affect the data stream used by the dowel search algorithm (see Fig. 3.55 and 3.59).

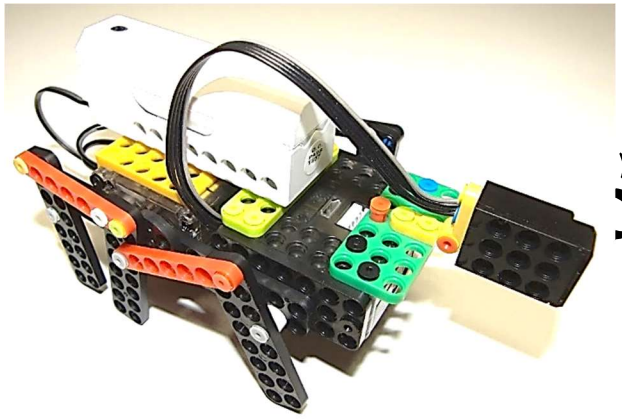

Fig. 6.13 Beetle Robot with NIR Dowel Scanning System.

For this Beetle project, the wanted operations for the robot are as follows:

1) The test wooden dowels need to be placed at random in front and to the right of the robot.

2) At start, the robot waits for the user's handclaps which correspond to the number of dowels that the robot is supposed to find.

3) Next, the robot swings right to find and approach the first dowel. When it gets close enough to the first dowel, it stops, plays a melody and waits for the user to clear the first dowel. Then it repeats the previous actions for the remaining dowels (one at a time).

The program "ScanningBeetle_NIR.tskx" was created to implement the above robot behaviors and it was derived from the "ScanningAvoider_CB.tskx" code. The algorithm used is described below:

1) For the first part of the Init Event, the robot plays Re# for 1 second (see Fig. 6.14 – lines 7-9), initializes various parameters (lines 10-17) and then waits for the user's handclaps (lines 18-19).

6.5 Racoon with PIR

The Passive IR Sensor (PIR-10) is a Motion Detector with a range of about 2 meters and a view angle of ± 45 degrees. In this application, it is mounted on the frame of a Racoon robot (see Fig. 6.21). Its output is "1" when motion is detected, and "0" when no motion is detected.

6.6 Bull Fight using NIR and Color Sensors

The Bull/Buffalo Robot has its assembly instructions in the R+DESIGN tool under the DREAM I folder and in the Paper Manual that came with the DREAM II Level 2 kit. **The reader should also note that, for**

the Bull Robot, the R+DESIGN's instructions mount the drive motors in reverse from the Paper Manual's instructions (at least currently in Spring 2018, the R+DESIGN information may be updated by ROBOTIS later). Thus the author created two versions of the Bull_Color codes: "Bull_Color_D.tskx" for R+DESIGN's users, and "Bull_Color_M.tskx" for Paper Manual's users.

The original Bull/Buffalo Robot uses the built-in Left and Right IR Sensors to detect and track an object, also it would charge forward when the detected object gets too close. This "revised" project additionally uses a Color Sensor (see Fig. 6.23) and illustrates a dual-range object detection scheme whereas:

- The NIR Sensors are used for the farther range (i.e. just to detect the presence of the object) and to make the robot move closer to the object.
- When the Color Sensor gets in range, then it is used to decide whether the object has a Red color or not. If a Red color is found, the Bull robot would back up and then charge ahead. For any other color found, the robot just plays a Melody.

The application code "Bull_Color_D.tskx" is shown in Fig. 6.24 which consists of a Main Endless Loop whereas:

- If the robot detects an object on its Left, it would turn left to track it (lines 20-21).
- As it gets closer to the object, the Left and Right IR Sensors would increase in value, and when both are larger than 100, the robot would next check the Color Sensor on Port 4 via the CALLBACK Function (i.e. get the latest Parameter CB_Color) (lines 23-25).
- If (CB_Color == Red) is TRUE, Function Charge is called (line 27), otherwise the robot would stop and Melody 21 is played (lines 31-34).
- Lines 38-55 (not shown) deal with the situation when an object is detected on the robot's Right in a similar manner.

Video 6.6 shows the run-time performance of the program "Bull_Color_D.tskx".

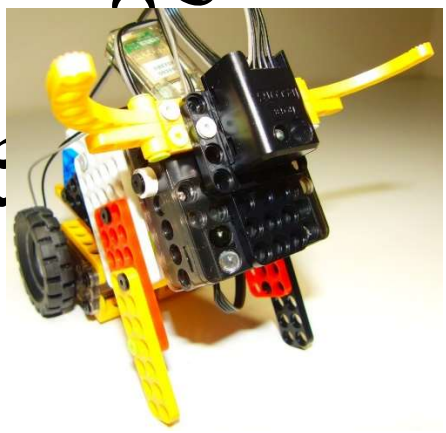

Fig. 6.23 Bull Robot with Color Sensor facing forward.

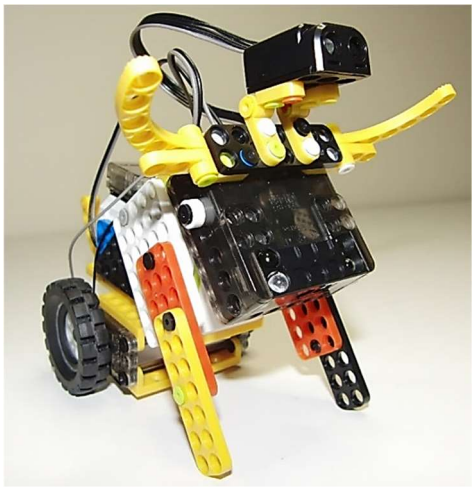

Fig. 6.25 The "Raging Bull" Robot.

6.8 MotoCycle

The MotoCycle Robot is newly introduced with the DREAM II School Set (see Fig. 6.34). It has a wide rear-drive wheel design (triple-tires) so that it can stand upright even when stopped (of course, not like a "real" motocycle). For this book's version, it has an additional external NIR Sensor (Port 4) mounted on the steering column to help it detect frontal obstacles when it is driven forward, while originally it only uses the built-in Center NIR Sensor to detect obstacles in its rear when it is driven backward. The steering column itself is controlled by a Servo Motor on Port 3.

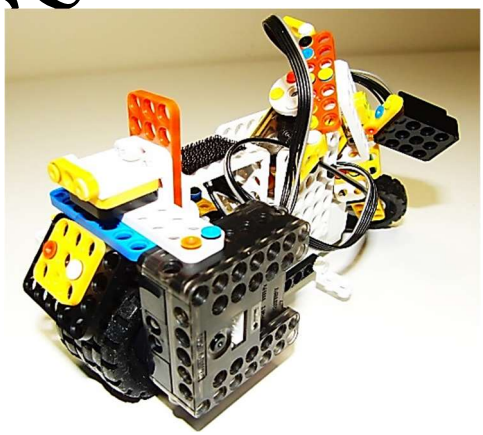

Fig. 6.34 "MotoCycle" with wide rear drive wheel and NIR Sensor mounted on steering column.

The "MotoCycle.tskx" has an "Init" event described in Fig. 6.35:

6.9 Four-Wheel Drive Dump Truck

This Dump Truck Robot is also a new example with the DREAM II system (see Fig. 6.42), in this book's version, it is enhanced with a rear-looking NIR Sensor which is used to trigger an Auto Dump procedure. It also features a 4 Wheel Drive concept using sprockets (see more details in Fig. 6.43). The actual Geared Motor's shaft is connected to the larger central sprocket which is engaged to 2 smaller sprockets which are connected to the actual wheels. Due this gear ratio transfer, the wheels rotate at a higher speed that the Geared Motors that drive them, making the Dump Truck the fastest robot in the example robots that can be built with the DREAM II School Set. This Dump Truck also features a carrier platform that can be raised or lowered using a Servo Motor on Port 3.

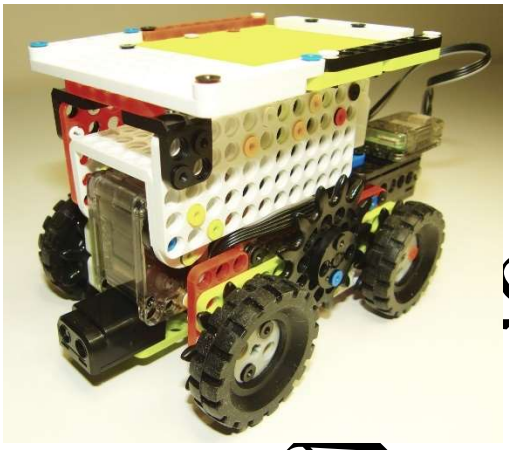

Fig. 6.42 4-Wheel Drive Dump Truck with rear NIR Sensor.