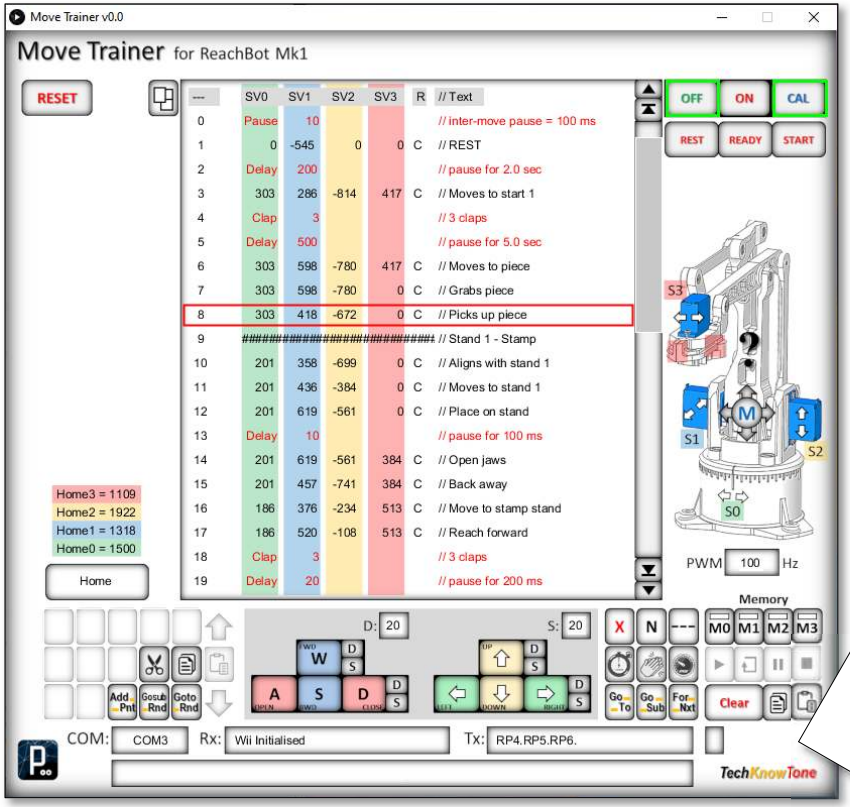
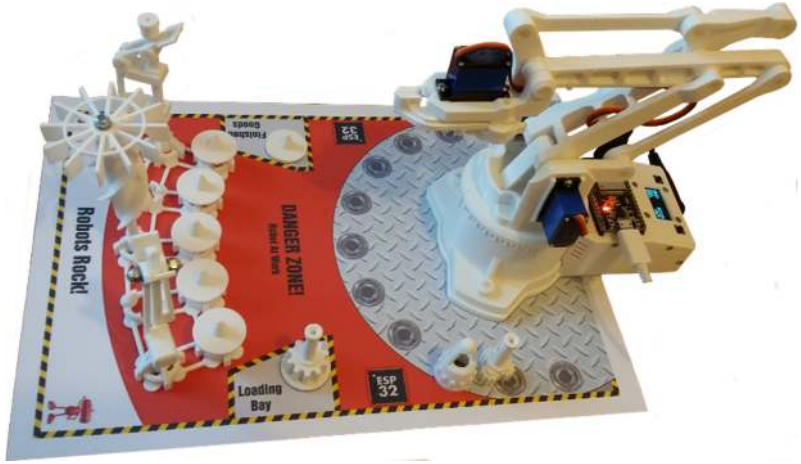
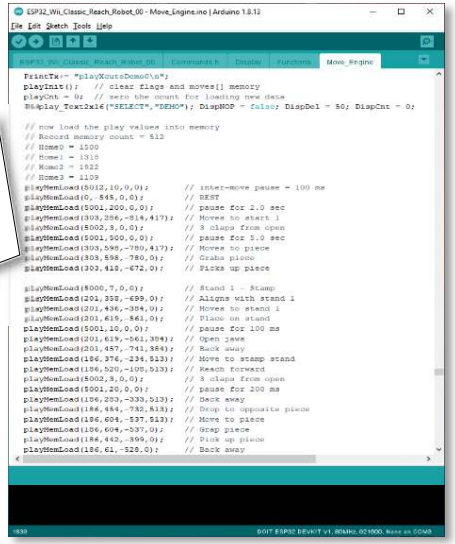
Reach Robot Mk1

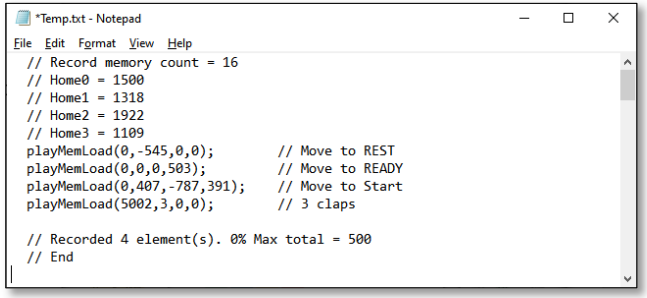
Programming App







```
PrintTx("playXoutoDemo5\n");
playInit();   // clear flags and moves[] memory
playCnt = 0;  // zero the count for loading new data
bool play_Text2x16("SELECT","DEMO"); DispNOP = false; DispDel = 50; DispCnt = 0;

// now load the play values into memory
// Record memory count = 512
// Home0 = 1500
// Home1 = 1318
// Home2 = 1922
// Home3 = 1109
playMemLoad(5012,10,0,0);       // inter-move pause = 100 ms
playMemLoad(0,-545,0,0);        // REST
playMemLoad(5001,200,0,0);      // pause for 2.0 sec
playMemLoad(303,286,-814,417);  // Moves to start 1
playMemLoad(5002,3,0,0);        // 3 claps from open
playMemLoad(5001,500,0,0);      // pause for 5.0 sec
playMemLoad(303,598,-780,417);  // Moves to piece
playMemLoad(303,598,-780,0);    // Grabs piece
playMemLoad(303,418,-672,0);    // Picks up piece

playMemLoad(8000,7,0,0);        // Stand 1 - Stamp
playMemLoad(201,358,-699,0);    // Aligns with stand 1
playMemLoad(201,436,-384,0);    // Moves to stand 1
playMemLoad(201,619,-561,0);    // Place on stand
playMemLoad(5001,10,0,0);       // pause for 100 ms
playMemLoad(201,619,-561,384);  // Open jaws
playMemLoad(201,457,-741,384);  // Back away
playMemLoad(186,376,-234,513);  // Move to stamp stand
playMemLoad(186,520,-108,513);  // Reach forward
playMemLoad(5002,3,0,0);        // 3 claps from open
playMemLoad(5001,20,0,0);       // pause for 200 ms
playMemLoad(186,283,-333,513);  // Back away
playMemLoad(184,484,-732,513);  // Drop to opposite piece
playMemLoad(186,604,-537,513);  // Move to piece
playMemLoad(186,604,-537,0);    // Grap piece
playMemLoad(186,462,-399,0);    // Pick up piece
playMemLoad(186,61,-528,0);     // Back away
```



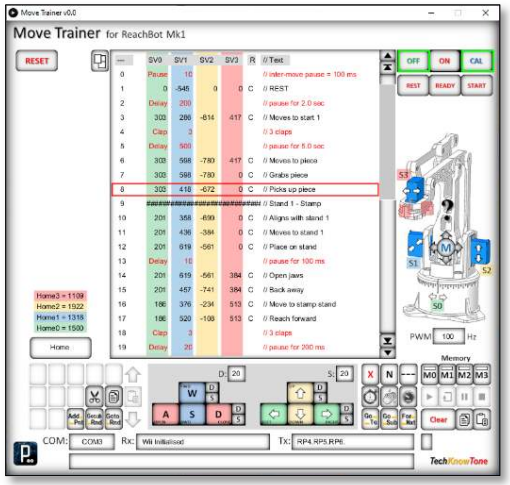
```
*Temp.txt - Notepad
File  Edit  Format  View  Help
  // Record memory count = 16
  // Home0 = 1500
  // Home1 = 1318
  // Home2 = 1922
  // Home3 = 1109
  playMemLoad(0,-545,0,0);        // Move to REST
  playMemLoad(0,0,0,503);         // Move to READY
  playMemLoad(0,407,-787,391);    // Move to Start
  playMemLoad(5002,3,0,0);        // 3 claps

  // Recorded 4 element(s). 0% Max total = 500
  // End
```

+ ESP32

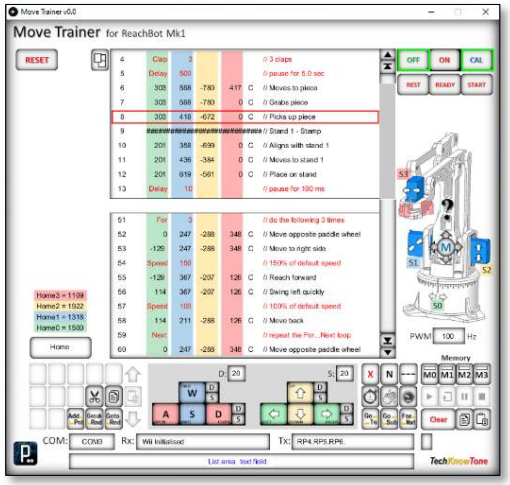
Issue: 1.0 Released: 18/03/2021 TechKnowTone

Introduction

Whilst the Reach Robot can be easily controlled via a Wii classic controller, in order to control and develop complex movements for a robot like the Reach Robot it was necessary to create a software application to run on a PC. The 'Move Trainer' app was written in C++ using a development environment called Processing. The IDE used in Processing was the forerunner to the Arduino IDE, so it provides a familiar interface to work with, and I recommend it to anyone wanting to create PC applications. See image bottom right of the Processing IDE.



When you invoke the app you are presented with the form shown top right. There is a graphical representation of the robot on the right-hand side, and a list window in the centre, which initially only contains one line of servo values. The image shown here is one of a complex move sequence loaded into the app. Start the app now whilst reading this.



Beneath the list window are a number of buttons, and communication fields; and the bottom field is an integrated help system. As you move the mouse over the form, help messages appear, to describe the function of the item beneath the mouse pointer. Give it a try.

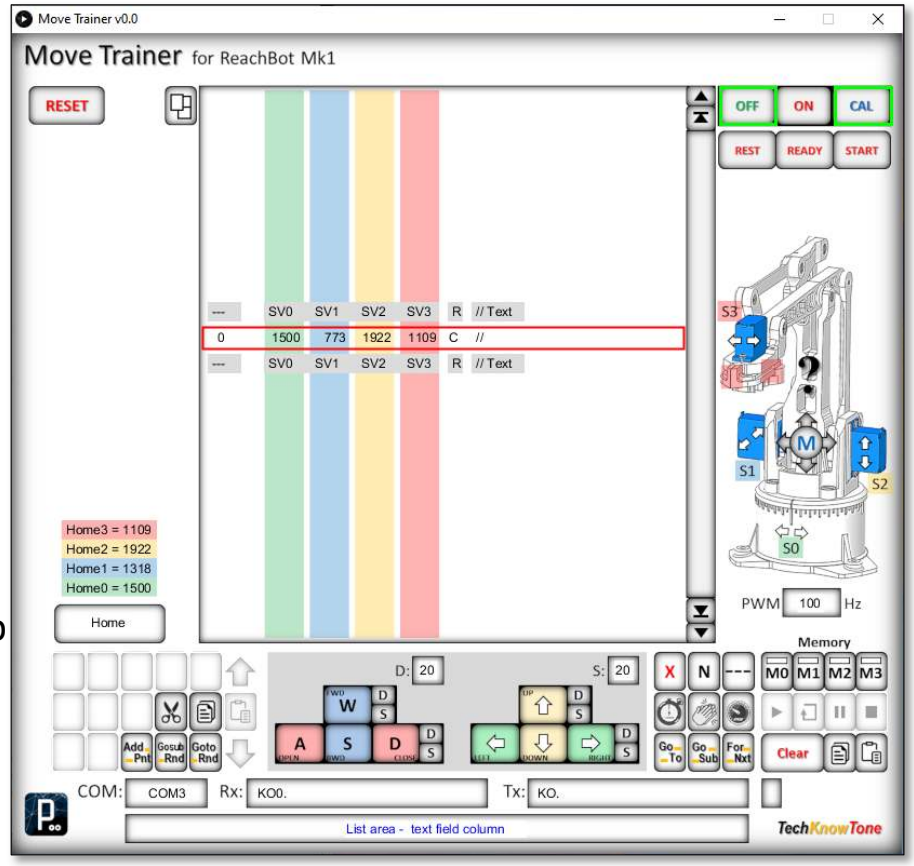
To use this Windows app you need to be connected to the robot via a USB serial link. During the process of establishing a connection between the robot and the app, the robot sends reference values to the app. Four of these values appear in the 'Home' fields on the left of the window. The app will not work normally until such values have been received from the robot.

Note that the app has a split window view of the list, to aid editing long lists, which can be toggled using the  button to the left of the list. All of the background graphics for this app were created using PowerPoint, and saved as a .png file.

Processing apps can be exported as both 32-bit and 64-bit executable files, and I have provided both with this release.

Move Trainer

The app provide different means by which you can adjust servo values, which are presented in four vertical colour coded columns. You can click on a number in the list, you can click and drag on the image of the robot, or you can enter adjustments using keyboard AWSD<>^v keys.



Once you have made the physical USB connection, you can connect the app to the robot by clicking on the COM: field, which displays – NA – initially in red. If the app is able to connect, it will display the Windows COM number in black. Note that a left mouse click invokes a USB connection, and a right click breaks a connection. This is useful to know, as you may want to upload code to you robot from the IDE, and if the app is connected to the serial port the IDE upload will fail.

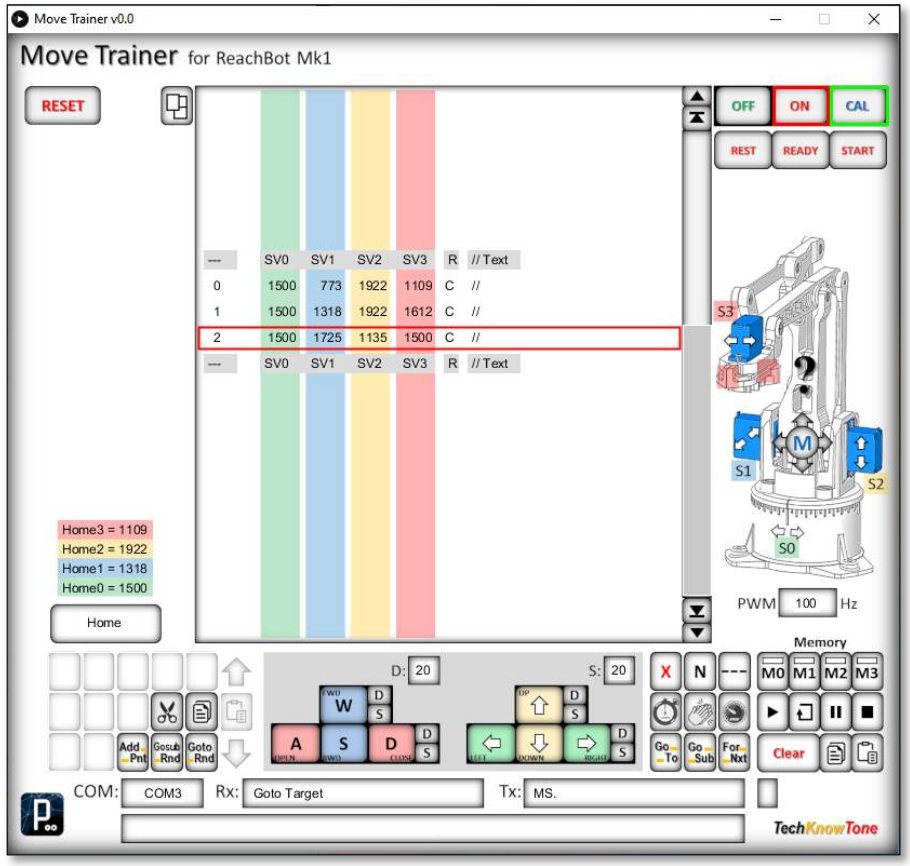
Whilst servo PWM values normally range from 1000 – 2000s, in this system we store values relative to their 'Home' positions, which prevents changes being needed should a servo fail, and need to be replaced.

Having made the connection to the robot you are now in a position to control it, but by default the robot is set to a '**SAFE**' condition, and this is indicated by the green box drawn around the **OFF** button. This allows for changes to be made to the list of servo values without the robot reacting to them, which could cause significant problems. To make the robot '**LIVE**' you simply click the **ON** button and the box drawn around it will go red if the robot is connected.

By default the robot will normally be in the **REST** position. If the servos are **ON**, then clicking on the **READY** button will move the robot to the 'Home' position, and the values in the active line of the list will change to suit. Now click on the **START** button and watch the robot move forwards and down. Each time you click one of these buttons, the robot moves to that position and the list changes. With this knowledge we will now create a simple movement…

A Simple ‘Move’ Program

With the servos active **ON** once again click on the **REST** button. Then click on the ‘**N**’ button, to insert a new line. Note that the new line becomes the active line, and its values are set to the previous line. Now click on the **READY** button to move the robot to that position and store the new values. Again click on the ‘**N**’ button toe insert a new line, which is a copy of the ready position, and finally click on the **START** button to set the servo values for that position. You should now have a list similar to the one on the right, but your servo values will be a little different.



You can now ‘Play’ this sequence of movements once, by clicking on the play button, or multiple times by clicking on the repeat button, until you stop the play process. Or you can scroll through the list with the mouse wheel or arrow keys to achieve the same effect.

Now scroll the list to make the 2nd line active (READY) and then click and drag the multi-move button in the centre of the robot image.

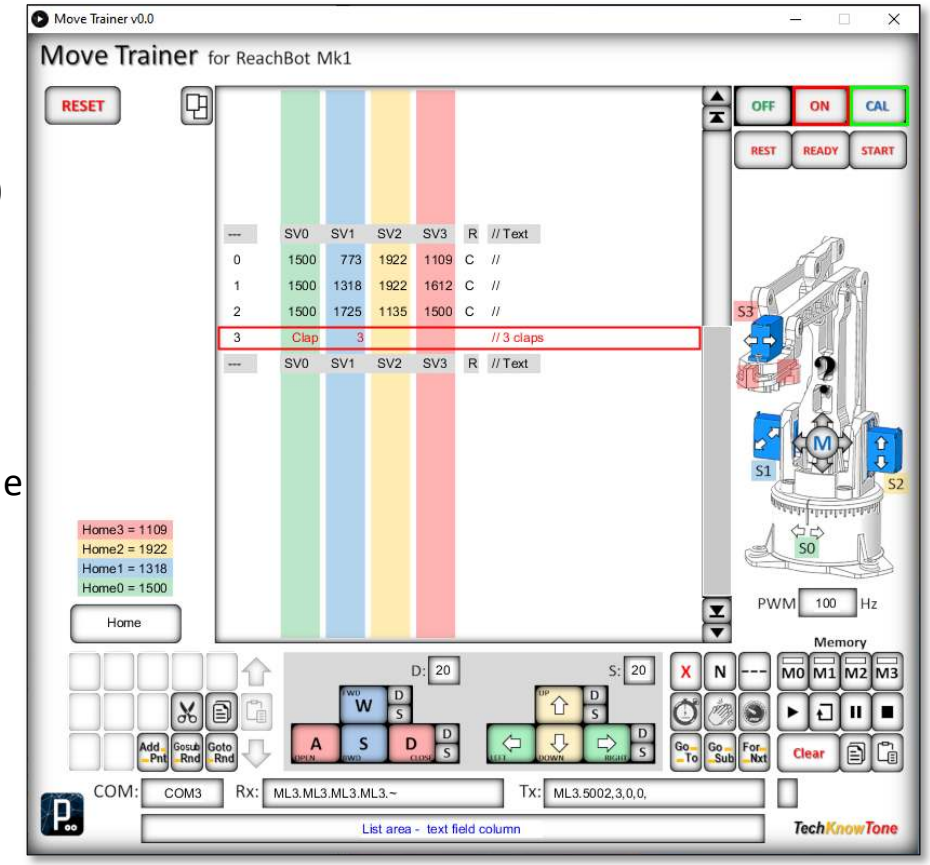
The robot should respond to your mouse movements by moving both servo S1 and servo S2 respectively, and the list values will change whilst you do this. Similarly you can click and drag on a single servo and change its value by moving the mouse in the direction of there associated arrow icons.

You now know how to change row values in the list and add new lines. A left click on the ‘**N**’ button creates a new line after the active line, and clicking with the right mouse button will create a new line before the active line. So you can now experiment and create a series of movements, play them and single step through them, forwards and backwards. Give it a try.

You can also select one or more lines, by clicking in the left-most column (line number), which gives you the ability to cut, copy and paste groups of lines, as well as move them up/down the list with the arrow buttons , which appear when lines are selected. We will next look at special functions….

Special Functions

We have seen that a list of movements can be created, with each row consisting of four servo values, S0 – S3. If the value received for servo S0 is >= 5000, then the robot considers that row to be a command line, and not simply servo values. To see how that works we will insert a ‘clap’ command. Scroll the list so that the last line is the active one, and then click on the clap button, beneath the ‘N’ button. The app will now insert a clap command into the list; commands are listed in red font. In the blue column (S1) the number will be 1, but you can click on that value to change it to 3.



Now when you play the sequence, when the robot gets to the last line it should hold its current position, but clap its jaws 3 times. The clap command ranges from 1 – 10, and in this example we have simply set servo S0 = 5002, and S1 = 3, but we don’t see those values as the app automatically presents them as commands in the list. If we click on the ‘Clap’ word in the S0 column, it will also toggle to ClapRnd.

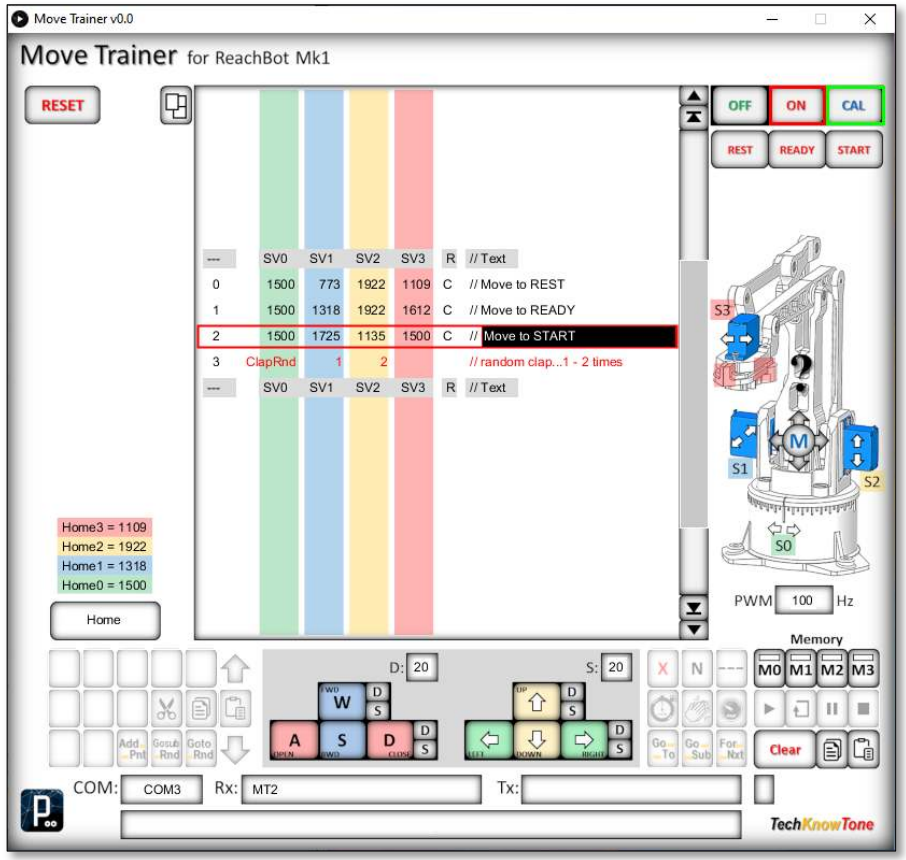
The clap command has now changed value to 5015, which means that it is a random clap ranging from values S1 to S2, and the S2 value now appears in the list, and can be changed by clicking on it, as before.

This simple, yet powerful system, enables you to insert other commands like delays and changes in speed for example, where ‘Delay’ can be toggled between random delays and inter-move pauses, giving the robot a stepped movement, rather than fluid one.

You will see that there are also buttons for inserting functions like For… Next loops, Goto and Gosub branch instructions, and you can even include ‘Point’ registers, preloaded with ‘AddPnt’ vectors that allow sequences to indirectly branch to different parts of the list in a random fashion, using GotoRnd and GosubRnd commands.

Make your comments

As you start to develop longer and longer movement sequences, which may include Gosub… Return branches it is important to include short text comments to remind you what is happening. In the image to the right you will see that I have included some comments in the first two rows, and I'm currently editing the 3rd row.



To start editing text you simple click in that field. The text editor code is simple, but it does support delete and cursor movements with the arrow keys. Pressing the keyboard RETURN key moves you onto then next line, whilst saving the line being edited. Pressing the ESC key exits the editor without saving the latest changes. So an editing session normally ends with the RETURN key (save), followed by the ESC key (exit).

If there are command in the list, the text editor will simply jump over them. You can also move up and down the rows by pressing the arrow keys, whilst remaining in text edit mode.

No UNDO

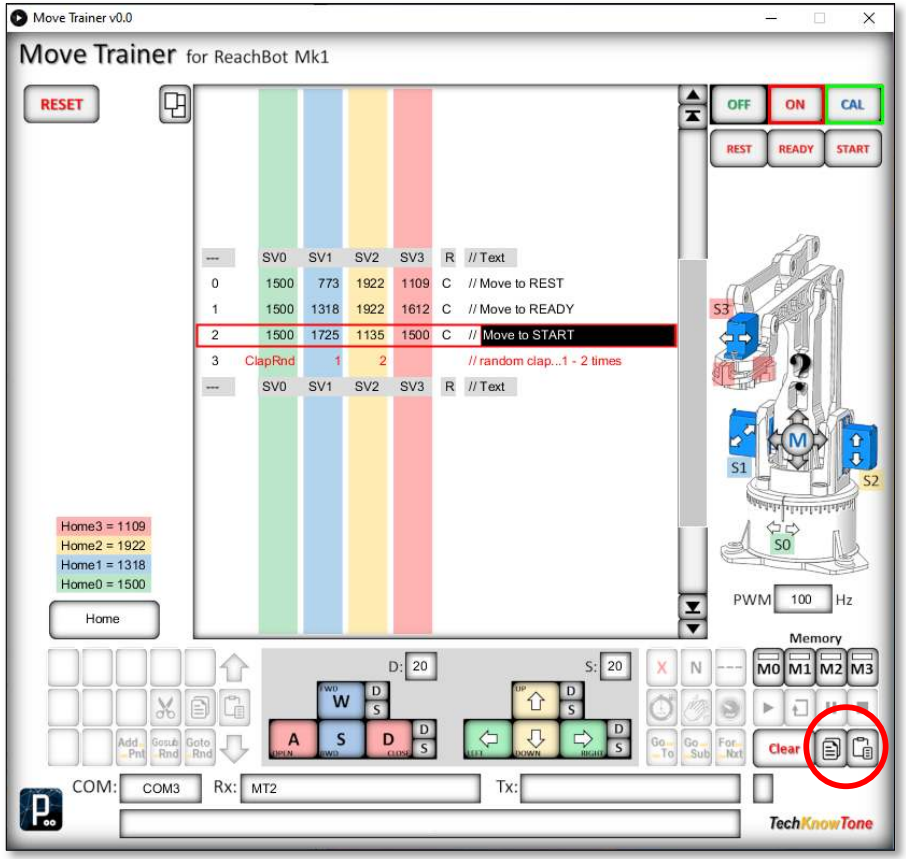
To save on complexity, a common feature not supported in this app is UNDO. So you do need to be carful when making changes, as you can effectively overwrite things that you might want to keep. What you can do however is that you can **ESC**ape out of a mouse movement, returning the robots servos to their starting positions. And you can place a copy of the whole of your list into one of three temporary stores, and recall them at any time. However the memory stores do not store your text comments, so they can be lost, and the servo data is lost once the app is terminated.

LIMITATIONS

At present the app and robot store up to 500 movements (list rows), which should be enough for most sequences; but the is scope to significantly increase that value if needed. Get in touch if that is the case.

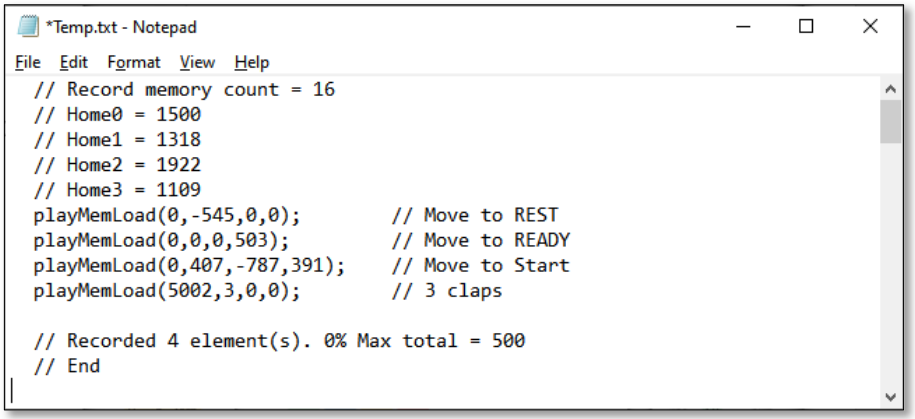
EXPORT & IMPORT

Whilst the Move Trainer does not save data to disc, it does have the ability to export and import data using the Windows clipboard. In doing so it treats the data as if it were being used in a .ino file, using the function playMemLoad(). To see this, we will open an empty file using the Windows Notepad app and copy data into it. To copy data onto the clipboard click on the copy button lower left, next to **Clear**.



If you then perform a paste operation within Notepad you should see the data as presented here. Note that the Home reference values are also exported at the start, and that servo angles are normalised to the Home values. Also note the 5002 code for the clap command. You would normally paste this code directly into a function within the Arduino IDE, but you can also save it to disc from Notepad if you wish.

This process also works in reverse, so you can take date from your .ino file by copying it from the IDE, or from a saved Notepad file. To see this, click on the Clear button to reset the list, then select the playMemLoad() items in Notepad and copy them. Then click and hold the paste button next to the **Clear** button. After a short delay your data should then appear in the Mover Trainers list.



You can test this further by going to the Move_Engine tab of the IDE and finding a function containing playMemLoad() data, and cut-n-paste it directly from there.

I hope this introduction has been helpful.