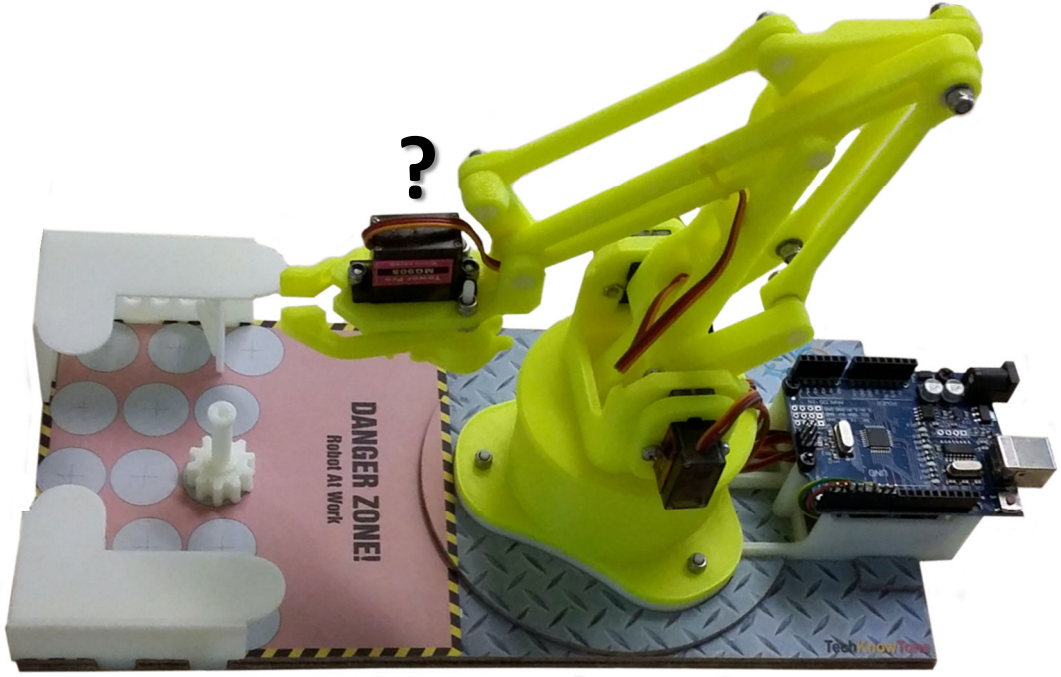
Reach Robot

Programming



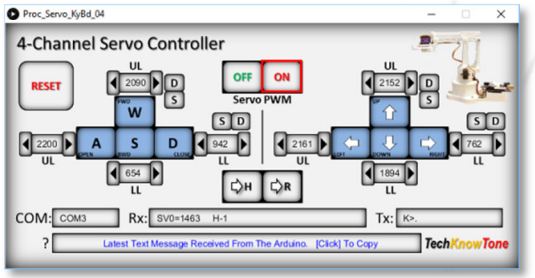
What do robots need to function?

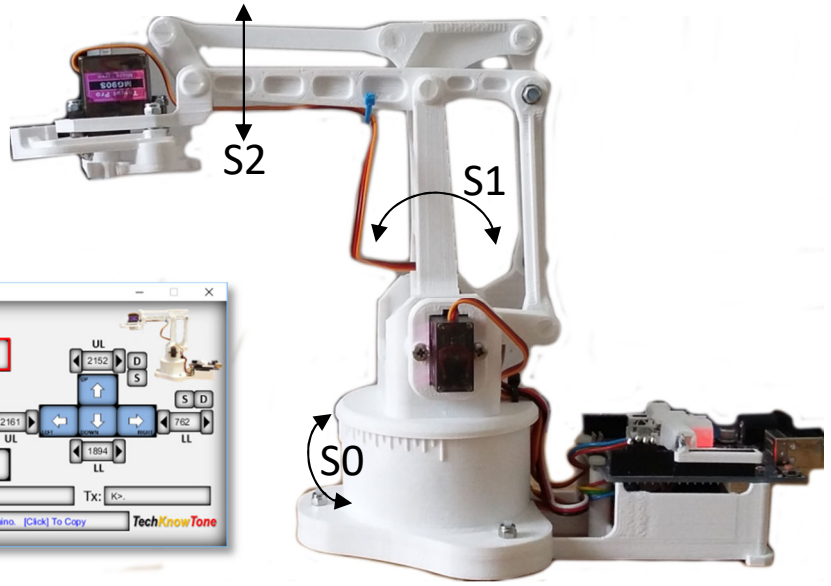
- Relative coordinates, XYZ from a datum
- Movement sequence, like X0,Y0,Z0 to X1,Y1,Z1, etc
- Movement speeds
- Time delays
- Special functions, like jaw movements
- Sensor inputs
- Stored in a repeatable CNC machine program

A robot needs to be taught these things!

Your Reach Robot needs the same information

- But it has limited program space
- We embed the program in the Arduino sketch
- Including servo calibration values
- Keyboard app helps us program it





Arduino Calibration Code Values:

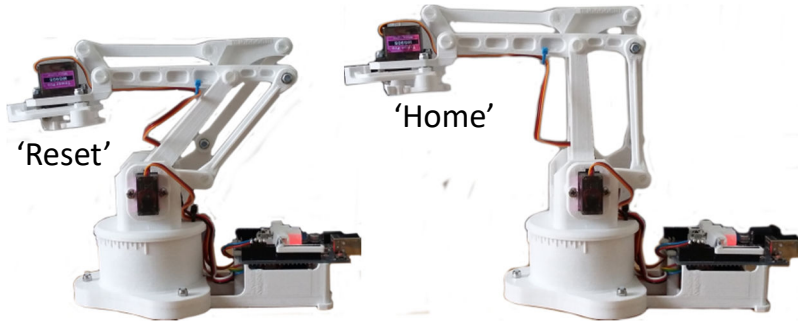
- We use servo PWM values as constants in the Arduino code:

```
// Define servo calibration constants
#define fwdArmMax 2276 // forward arm Max servo value
#define fwdArmMin 1006 // forward arm Min servo value
#define fwdArmVert 1481 // forward arm vertical servo value
#define gripClose 962 // jaws closed servo value
#define gripOpen 1360 // jaws moderately open value (23%)
#define gripWide 2200 // jaws wide open value
#define turntableCtr 1461 // turntable servo centre value
#define turntableMax 2161 // turntable servo Max value
#define turntableMin 762 // turntable servo Min value
#define vertArmMaxA 1907 // vertical arm Max 'A' servo value
#define vertArmMaxB 2151 // vertical arm Max 'B' servo value
#define vertArmMinA 951 // vertical arm Max 'A' servo value
#define vertArmMinB 1333 // vertical arm Max 'B' servo value
#define vertArmMinC 1894 // vertical arm Max 'C' servo value
```

Values determined during the 'Fine' calibration process and entered into the code as constants.

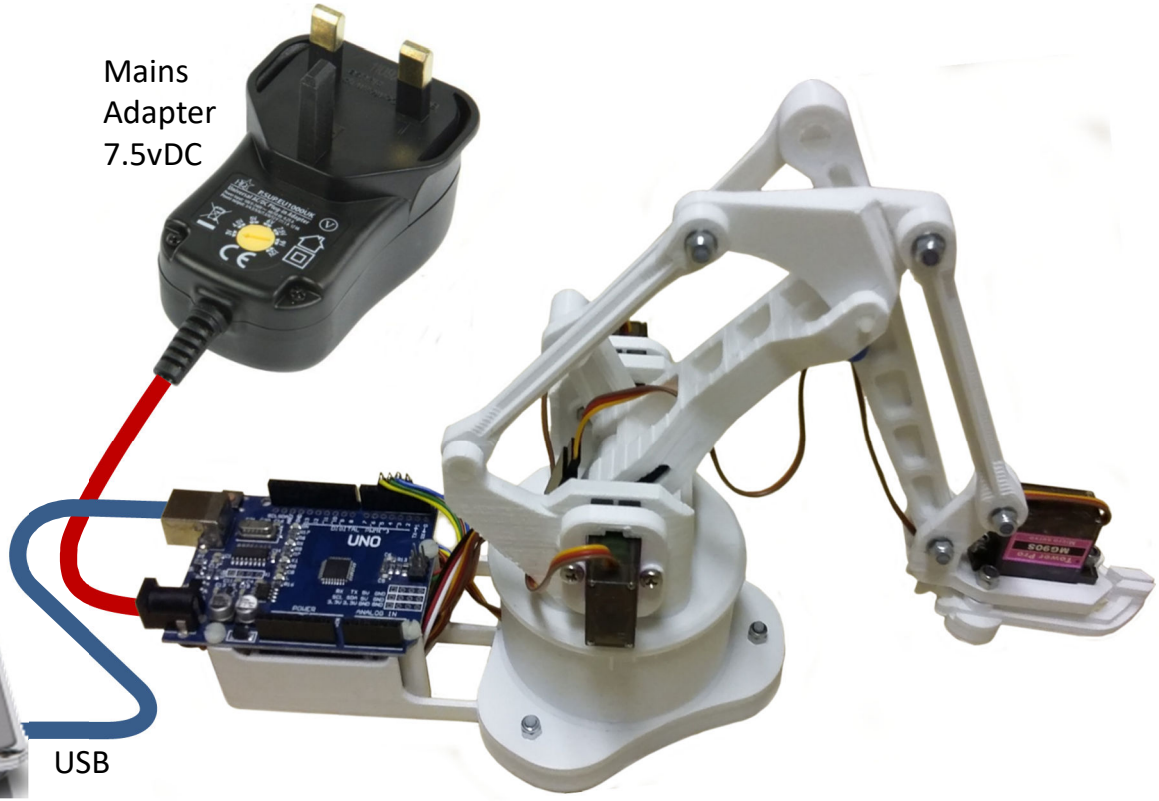
```
#define Home0 turntableCtr // home position for servo 0
#define Home1 1500 // home position for servo 1
#define Home2 1900 // home position for servo 2
#define Home3 gripOpen // home position for servo 3
#define Reset0 turntableCtr // RESET position for servo 0
#define Reset1 1500 // RESET position for servo 1
#define Reset2 1900 // RESET position for servo 2
#define Reset3 gripOpen // RESET position for servo 3
#define servoOffMax 44 // sets maximum thermal drift offset for servo 0
#define servoOffRmpDwn 60000 // sets thermal offset ramp down time in miliseconds
#define servoOffRmpUp 10000 // sets thermal offset ramps up time in miliseconds
```

Values determined after the 'Fine' calibration process. You will have determined your own 'Home' and 'Reset' co-ordinates using the Windows calibration app.



You will have determined your own calibration values for your Reach Robot and added them to your Arduino sketch.

Robot Programming:



Things needed for Robot Programming:

- Windows PC, with Java runtime installed
- Arduino IDE; latest version with necessary libraries
- Servo Keyboard app: Servo_KeyBrd_Cntrl.exe
- Your calibrated Arduino .ino files
  - Something like> My_Reach_Robot_00.ino
- An understanding of the 'Move' commands

Note: Your robot will need to be powered from a 7.5v mains adapter for the servo motors to function.

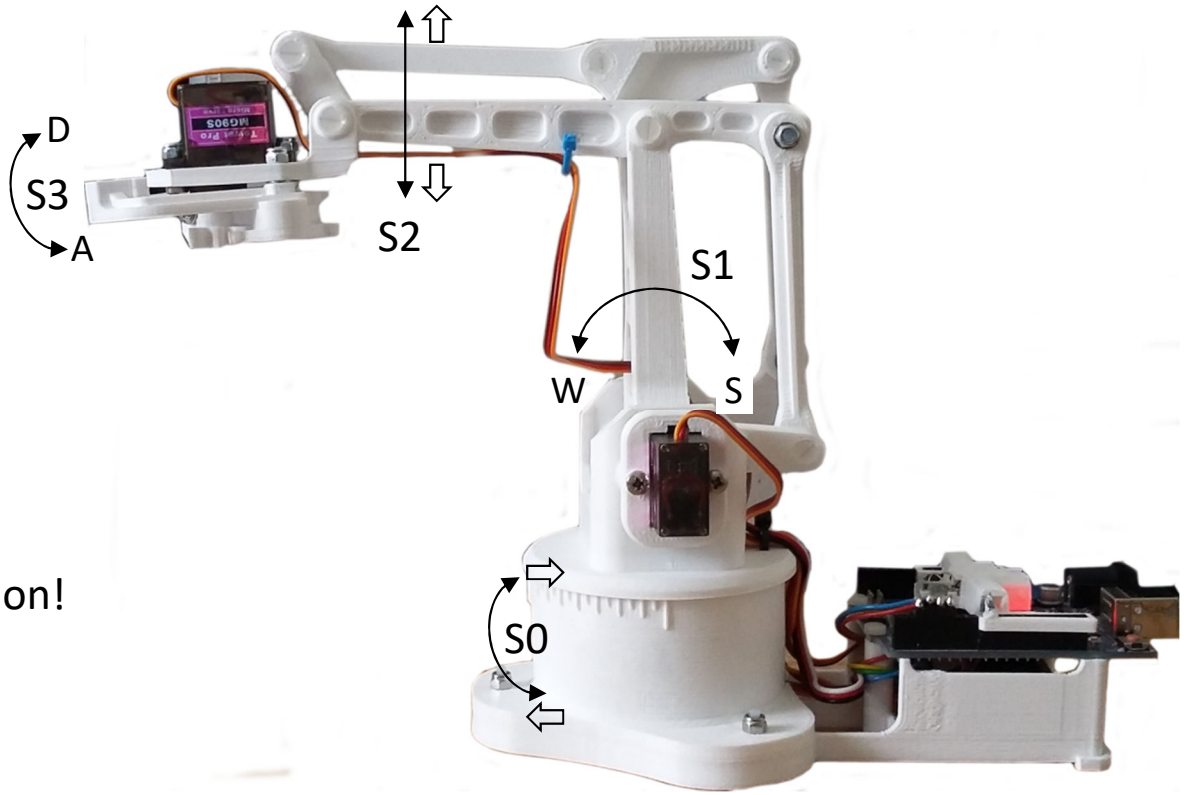
You can't use the IDE serial monitor or program the Arduino whilst the keyboard app is running, as it hogs the USB interface. But if you click-right on the COM field this disconnects the keyboard app from the USB port.

Reach Robot CNC Programs

An Arduino UNO sketch limits us to:

- Programs of 300 data blocks in length
- you can pre-store 32 XYZ locations
- you can use 10 labels for branching
- ‘Nesting’ is not supported

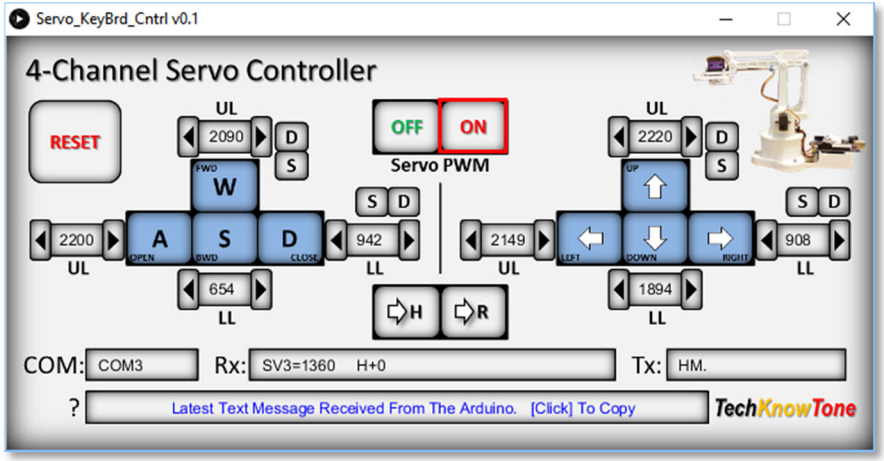
An Arduino MEGA would improve these limitation!



Servo Keyboard Control App

You will use this Windows App to:

- Move the robot servos to position the head
- Extract servo values for use in the CNC program
- Load and execute your program
- Confirm data block limits are not exceeded
- Fine tune positions

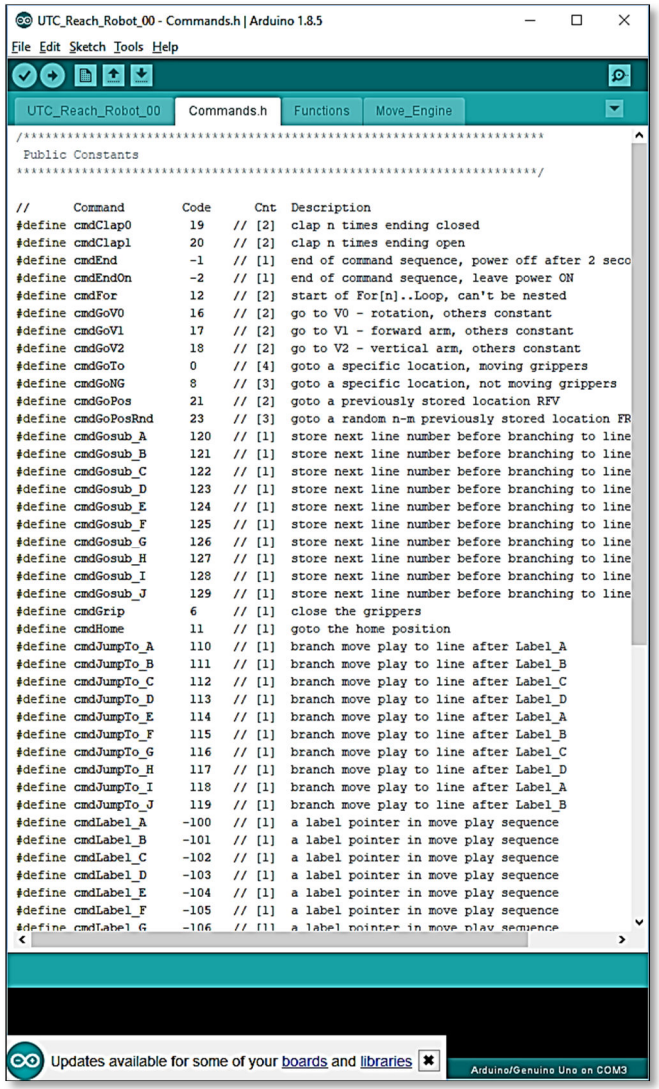


CNC Program ‘Commands’

‘Move Engine’ commands are declared and listed in Commands.h
This text file is automatically loaded with your Arduino sketch

```
/*************************************************************************************
 Public Constants
*************************************************************************************/

//      Command         Code      Cnt  Description
#define cmdClap0         19    // [2]  clap n times ending closed
#define cmdClap1         20    // [2]  clap n times ending open
#define cmdEnd           -1    // [1]  end of command sequence, power off after 2 seconds
#define cmdEndOn         -2    // [1]  end of command sequence, leave power ON
#define cmdFor           12    // [2]  start of For[n]..Loop, can't be nested
#define cmdGoV0          16    // [2]  go to V0 - rotation, others constant
#define cmdGoV1          17    // [2]  go to V1 - forward arm, others constant
#define cmdGoV2          18    // [2]  go to V2 - vertical arm, others constant
#define cmdGoTo          0     // [4]  goto a specific location, moving grippers
#define cmdGoNG          8     // [3]  goto a specific location, not moving grippers
#define cmdGoPos         21    // [2]  goto a previously stored location RFV
```



All command mnemonics start with the ‘cmd’ prefix, like cmdClap0.
It is important to realised that commands may require a different number of memory blocks (Cnt).
Command mnemonics are actually stored as numbers (codes) in a memory array. ie. cmdEnd == -1
We only need to use the mnemonic and the block count to load them into memory.

We develop our program within a sketch ‘Function’

```
void moveToTest00 () {
  // a move to test sequence with 3 fixed positions
  Serial.println(F("Loading moveToTest00..."));
  moveInit(); // always Initialise Move Engine before loading arrays!
  movePause = 50; // set delay aftyer move to 50 loops, default = 0

  // load target positions
  moveLoadPosRFV(0, Home0-302, Home1+731, Home2-835); // loading bay, backed off
  moveLoadPosRFV(1, Home0-302, Home1+696, Home2-771); // loading bay, pick up
  moveLoadPosRFV(2, Home0, Home1+217, Home2-642); // centre back, mid height
  moveLoadPosRFV(3, Home0+425, Home1+374, Home2-701); // finishing bay, backed off
  moveLoadPosRFV(4, Home0+425, Home1+696, Home2-771); // finishing bay, drop off
  moveLoadPosRFV(5, Home0-172, Home1+744, Home2-598); // RH low stand
  moveLoadPosRFV(6, Home0-172, Home1+368, Home2-598); // RH low stand, backed off
  moveLoadPosRFV(7, Home0+40, Home1+654, Home2-560); // Ctr mid stand
  moveLoadPosRFV(8, Home0+40, Home1+368, Home2-598); // Ctr mid stand, backed off
  moveLoadPosRFV(9, Home0+260, Home1+586, Home2-512); // LH high stand
  moveLoadPosRFV(10, Home0+260, Home1+378, Home2-530); // LH high stand, backed off
  moveLoadPosRFV(11, Home0-180, Home1+443, Home2-191); // above press leaver
  moveLoadPosRFV(12, Home0-180, Home1+494, Home2-336); // press leaver down
  moveLoadPosRFV(13, Home0-137, Home1+371, Home2-362); // move to right of spin wheel
  moveLoadPosRFV(14, Home0+207, Home1+371, Home2-362); // move to the left of spin wheel
  moveLoadPosRFV(15, Home0+242, Home1+346, Home2-140); // opposite flag pole
  moveLoadPosRFV(16, Home0+242, Home1+572, Home2-140); // opposite flag pole

  // load move sequence
  moveLoad1(cmdHome); // start at home position
  moveLoad2(cmdGoPos, 0); // loading bay, backed off
  moveLoad2(cmdGoPos, 1); // loading bay, pick up

  moveLoad2(cmdClap1, 5); // clap 5 times to get attention!
  moveLoad2(cmdWait, 450); // wait for 5 seconds to load object
  moveLoad1(cmdGrip); // close jaws to collect object

  // move object to 1st stand
  moveLoad1(cmdGosub_A); // call common subroutine _A

  // place the object on the 1st stand
  moveLoad2(cmdGoPos, 5); // RH low stand
  moveLoad1(cmdOpen); // open jaws to drop off object
  moveLoad2(cmdGoPos, 6); // RH low stand, backed off
```

Sketch function name: moveToTest00
Initialisation calls and settings go here.

We load up to 32 target positions into an array.
It is best to use relative values, like Home0 - 302,
but servo PWM numbers can be used instead.

Always use // descriptive text to remind you what the location is you are defining. This will help greatly when you come to reference it later. ie. cmdGoPos, 5.

Defining target positions can save code blocks and allow the used of the random command, cmdGoPosRnd.

You next write lines of code to load the move commands into the program array. Note that the number associated with the moveLoad… function call relates to the number of values being added to the program. For example moveLoad2(cmdWait,450) loads two values into memory; ie. the code for cmdWait and the value 450.

Programming continued:

```
moveLoad2(cmdWait, 30); // wait for 3/10 seconds
moveLoad2(cmdSet3, Home3+386); // open jaws to release flags
moveLoad2(cmdWait, 30); // wait for 3/10 seconds
moveLoadl(cmdNext); // repeat For... loop
moveLoadl(cmdHome); // goto home

moveLoadl(cmdGosub_A); // call common subroutine _A

// collect the object from the 3rd stand
moveLoad2(cmdGoPos, 10); // LH high stand, backed off
moveLoad2(cmdGoPos, 9); // LH high stand
moveLoadl(cmdGrip); // close jaws to collect object

moveLoadl(cmdGosub_A); // call common subroutine _A

// place the object in the finishing bay
moveLoad2(cmdGoPos, 3); // finishing bay, backed off
moveLoad2(cmdGoPos, 4); // finishing bay, drop off
moveLoadl(cmdOpen); // open jaws to drop off object
moveLoad2(cmdGoPos, 3); // finishing bay, backed off
moveLoad2(cmdWait, 50); // wait for 1 seconds

// retreat to reset and clap
moveLoadl(cmdGosub_A); // call common subroutine _A
moveLoadl(cmdReset); // goto reset position
moveLoad2(cmdClap0, 3); // clap 3 times to get attention!
moveLoadl(cmdEnd); // stop at th is point

// common subroutine
moveLoadl(cmdLabel_A); // give this subroutine a label
moveLoad2(cmdGoPos, 2); // centre back, mid height
moveLoad2(cmdWait, 50); // wait for 1 seconds
moveLoadl(cmdReturn); // return to line after Gosub

// report the amount of program space used
Serial.print(F("Loaded ")); Serial.print(movePnt);
Serial.print(F(" commands. ")); Serial.print((movePnt * 100)/moveArraySize);
Serial.print(F("% Max total = ")); Serial.println(moveArraySize);
}

// --------------------------------------------------------------------
```

More commands defining the robots moments.

Note the cmdGosub_A calls to the subroutine with the label name cmdLabel_A, and how the code sequence returns to the next line after reaching the cmdReturn command. Subroutines are used to reduce the number of code blocks used.

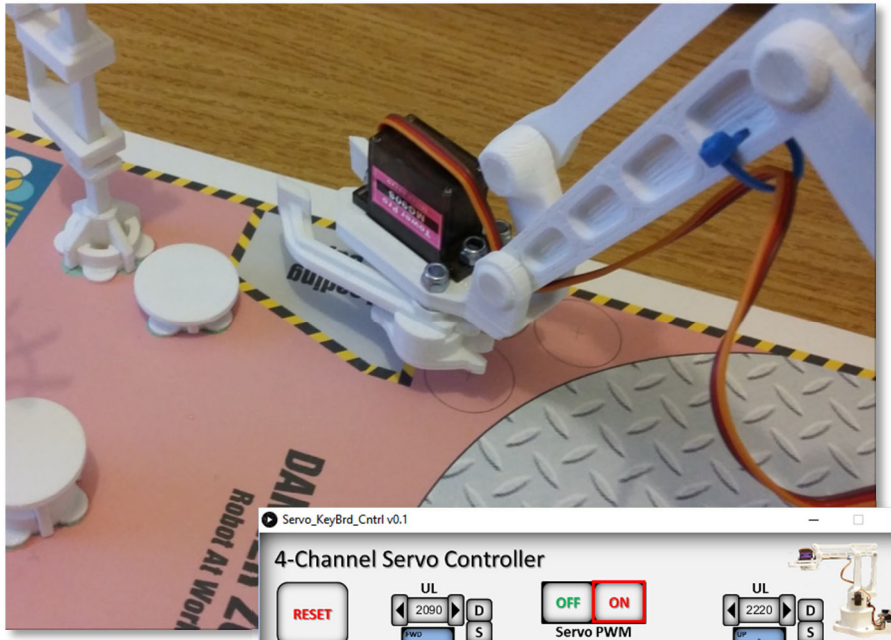
Note that the cmdEnd command is needed to avoid running into the subroutine code after cmdLabel_A.
If we don't use subroutines we don't need to used the end command.

Place subroutines at the end of your program, so that you don't have to jump over them.

Having loaded your commands into memory this sketch function reports the number of code blocks consumed by it. A useful check to ensure you don't exceed the limit!

Simple Example:

We will move the robot from the ‘Home’ position to a place on a board, where it will open and close its jaws 3 times.



- Use the Servo app to position the robot head.
- Tap the spacebar three times to get the readings:

Rx: moveLoadPosRFV(0, Home0-325, Home1+760, ...

- Click on this field to copy it to the clipboard.
- Past the copied text into your sketch and add the following lines so that it looks like this:

```
moveInit(); // always Initialise Move Engine before loading arrays!

// load target positions
moveLoadPosRFV(0, Home0-325, Home1+760, Home2-744); // your chosen location

// load move sequence
moveLoad1(cmdHome);     // start at home position
moveLoad2(cmdGoPos, 0); // goto location 0
moveLoad2(cmdClap1, 3); // clap 3 times to get attention!
moveLoad1(cmdEnd); // stop at this point
```

- Now compile your sketch and load it into the Arduino.
- Click-right on the app COM field to temporarily disconnect it.
- With sketch loaded, reconnect the COM link by clicking-left
- To execute the program click on the Tx field and type **MX0.**
- Then press the RETURN key to run it.

Tx: MX0.

Servo Keyboard App Features

RESET the app and the Arduino sketch.

Switch PWM OFF to save power. Keeps servos cooler.

PWM must be ON for many features to work. 2 second auto-cut-off.

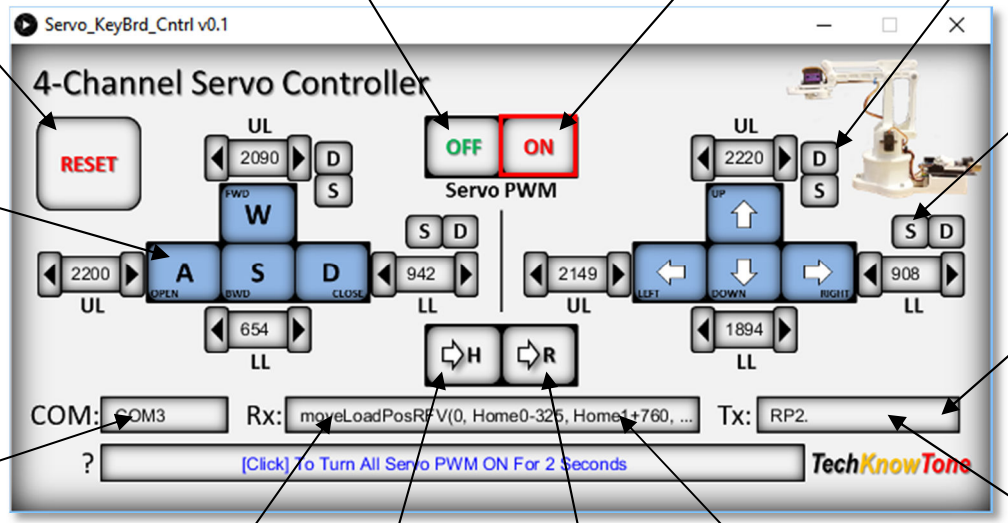
Dither helps with fine tuning head positions.

Flat topped sinusoidal sweep.

Press keyboard keys or click on the window icons.

Text commands sent from the app to the Arduino.

Click-left to connect via USB. Click-right to disconnect.



Click in this field to enter and send commands directly.

Tx: MX0.

Press SPACEBAR to request all servo values from the Arduino.

Go to the Home position. 30 second auto-cut-off

Go to the RESET position. 2 second auto-cut-off

Click this field to copy its contents to the clipboard.