uArm gCode Communication Protocol

Content

1.1 Brief description

UArm gCode is an important part of the uArm firmware 2.0. Based on the standard gCode protocol, we add a new protocol head in front of the gCode so that it can be more easily to use and debug. What's more it is designed to be compatible with the standard gCode. (We offer the code of decode the standard gCode)

1.2 Description of the protocol

Example:

sending command from PC "#25 G0 X12 Y23 Z51 F55\n" //move to [12,23,51] while keeping the speed at 55

Reply from uArm "$25 OK V500\n" //finish the command and return value = 500

Command can be divided into two parts:

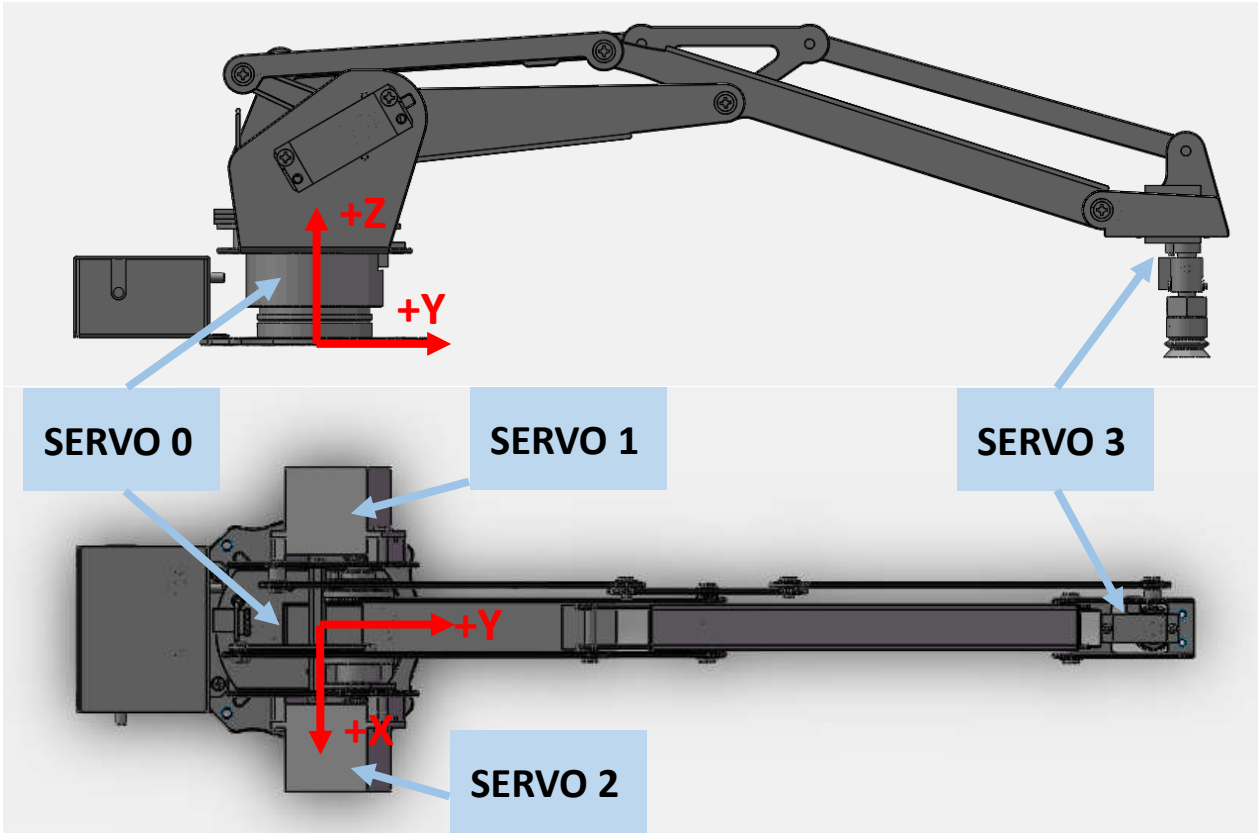
1. Command with underline: it's the new added protocol head. The command from PC starts with '#', while the command from uArm starts with'$'. And the data following the symbol decided by the PC, and the reply from the uArm should have the same data which indicates it finish the command. (In the example above, PC sends the command with '#25' and uArm replies the command with '$25')

2.Command without the underline: it's the standard gCode.

Caution:

1.Currently the communication protocol should contain both the command with underline and the command without underline; (Because the code space of uArm control board is not big enough)
2.There should be blank space between each parameter;
3.The letters in the command should be capitalized;
4.The command should end with '\n'.

1.3 Definition of the coordinates



1.4Commands

GCode Command	Description	Feedback
Moving Command (parameters are in underline)		
“#*n* G0 X*100* Y*100* Z*100* F*100*\n”	Move to XYZ(mm), F is speed(mm/min)	”$*n* OK \n” or ”$ *n* E*x* \n”(refer to Err output)
“#*n* G201 S*100* R*90* H*80* F*100*\n”	Polar coordinates, S is stretch(mm), R is rotation(degree),H is height(mm), F is speed(mm/min)	”$*n* OK \n” or ”$ *n* E*x* \n”(refer to Err output)
“#*n* G202 N*0* V*90*\n”	Move the servo to the position ,N is ID of servos(0~3),V is angle(0~180)	”$*n* OK \n” or ”$ *n* E*x* \n”(refer to Err output)
“#*n* G203\n”	Stop moving	”$*n* OK \n” or ”$ *n* E*x* \n”(refer to Err output)
“#*n* G204 X*10* Y*10* Z*10* F*100*\n”	relative displacement	”$*n* OK \n” or ”$ *n* E*x* \n”(refer to Err output)
Setting Command (parameters are in underline)		
“#*n* M120 V*0.2*\n”	Set time cycle of feedback, return Cartesian coordinates, V is time(seconds)	”@3 X*154.714* Y*194.915* Z*10.217*\n”
“#*n* M200\n”	Check if uArm is moving	”$*n* OK V*1*\n” (1 moving,0 stop)
“#*n* M201 N*0*\n”	attach servo, N is ID of servos(0~3)	”$*n* OK \n” or ”$ *n* E*x* \n”(refer to Err output)
“#*n* M202 N*0*\n”	Detach servo, N is ID of servos(0~3)	”$*n* OK \n” or ”$ *n* E*x* \n”(refer to Err output)
“#*n* M210 F*1000* T*0.2*\n”	buzzer,F is frequency, T is time (s)	”$*n* OK \n” or ”$ *n* E*x* \n”(refer to Err output)
“#*n* M211 N*0* A*200* T*1*\n”	Read EEPROM N(0~2,0 is internal EEPROM,1 is USR_E2PROM, 2 is SYS_E2PROM), A is address, T is type (1 char,2 int,4 float)	”$*n* OK V*10*\n”
“#*n* M212 N*0* A*200* T*1* V*10*\n”	Write EEPROM N(0~2,0 is internal EEPROM,1 is USR_E2PROM, 2 is SYS_E2PROM), A is address, T is type (1 char,2 int,4 float)V is the input data	”$*n* OK\n”
“#*n* M220 X*100* Y*100* Z*100*\n”	Convert coordinates to angle of servos	”$*n* OK B*50* L*50* R*50*\n” (B servo 0,L

		servo1,R servo 2, 0~180)
“#n M221 B0 L50 R50\n”	Convert angle of servos to coordinates	”$n OK X100 Y100 Z100\n”
“#n M222 X100 Y100 Z100 P0\n”	Check if it can reach,P1 polar, P0 Cartesian coordinates	”$n OK V1\n” (1 reachable, 0 unreachable)
“#n M231 V1\n”	pump V1 working, V0 stop	”$n OK \n” or ”$n Ex \n”(refer to Err output)
“#n M232 V1\n”	gripper V1 close, V0 open	”$n OK \n” or ”$n Ex \n”(refer to Err output)
“#n M240 N1 V1\n”	Set the digital IO output	”$n OK \n” or ”$ n Ex \n”(refer to Err output)
Querying Command (parameters are in underline)		
“#n P200\n”	Get the current angle of servo	”$n OK B50 L50 R50\n”
“#n P201\n”	Get the device name	”$n OK V3.2\n”
“#n P202\n”	Get the hardware version	” $n OK V1.2\n”
“#n P203\n”	Get the software version	” $n OK V3.2\n”
“#n P204\n”	Get the API version	” $n OK V3.2\n”
“#n P205\n”	Get the UID	” $n OK V3.2\n”
“#n P220\n”	Get current coordinates	” $n OK X100 Y100 Z100\n”
“#n P221\n”	Get current polar coordinates	” $n OK S100 R90 H80\n”
“#n P231\n”	Get the status of pump	” $n OK V1\n” (0 stop, 1 working, 2 grabbing things)
“#n P232\n”	Get the status of gripper	” $n OK V1\n” (0 stop, 1 working, 2 grabbing things)
“#n P233\n”	Get the status of limited switch	“$n OK V1”(R1 triggered, R0 untriggered)
“#n P240 N1\n”	Get the status of digital IO	” $n OK V1\n” (1 High, 0 Low)
“#n P241 N1\n”	Get the status of analog IO	” $n OK V295\n” (return the data of ADC)
Ticking feedback		
@3	Timed feedback , ”M120”	
Err Output		
E20	Unexist command	
E21	Parameter error	
E22	Address out of range	

1.5 Example code of decoding gCode

```
#define MAX_CMD_SIZE        960
#define BUFSIZE             4

typedef struct
{
    char cmdbuffer[BUFSIZE][MAX_CMD_SIZE];
    float value_V;
    int buflen;
    int bufindr;
    int bufindw;
}gc_Command_t;
gc_Command_t gc_Command;

int example_test()
{
    long temp_l;
    char ch;
    uint8_t command_num;

    if(code_seen('#'))
    {
        command_num = (int)code_value();
    }
    else
    {
        //no “#n”， error return;
        return -1;
    }

    if (code_seen('G') && sys_state.online)
    {
        switch ((int)code_value())
        {
        case 1:
            get_commandValue();
            break;

        case 2:
            break;
        case 3:
            break;
        default:
            Printf(MYSERIAL, "unknow conmand\n");
        }
    }
    else if (code_seen('M'))
    {
        switch ((int)code_value())
        {
        case 0:
            kill_system();
            break;
        case 5:
            break;
        default:
            Printf(MYSERIAL, "unknow conmand ");
        }
        Printf(MYSERIAL,"\n");
    }
```

```
        return 1;
}

void get_commandLine()
{
        char serial_char;
        static long gcode_N, gcode_LastN;
        static bool comment_mode = false;
        static int serial_count = 0;

        while( gc_Command.buflen < BUFSIZE)
        {
                serial_char = usart1_receBuf_read();      //read a char
                if (0 == serial_char)
                {
                        break;
                }

                // get a complete command
                if (serial_char == '\n' ||serial_char == '\r' || (serial_char == ':' && comment_mode == false) ||
serial_count >= (MAX_CMD_SIZE - 1) )
                {
                        if (!serial_count)                              //if empty line
                        {
                                comment_mode = false;                   //for new command
                                return;
                        }

                        gc_Command.cmdbuffer[gc_Command.bufindw][serial_count] = 0;   //terminate string
                        if (!comment_mode)
                        {
                                comment_mode = false;                   //for new command

                                //if have 'N' and into check
                                if (strchr(gc_Command.cmdbuffer[gc_Command.bufindw], 'N') != NULL)
                                {
                                        //strchr function search string first appeared in the position of the character Char
                                        strchr_pointer = strchr(gc_Command.cmdbuffer[gc_Command.bufindw], 'N');

                                        //strtol function ,if "123asd" ,return long 123 ,10:decimalism
                                        gcode_N = (strtol(&gc_Command.cmdbuffer[gc_Command.bufindw][strchr_pointer -
gc_Command.cmdbuffer[gc_Command.bufindw] + 1], NULL, 10));

                                        //check
                                        if                (gcode_N                !=                gcode_LastN+1                &&
(strstr(gc_Command.cmdbuffer[gc_Command.bufindw], "M110") == NULL) )
                                        {
                                                serial_count = 0;         //clear buffer
                                                return;
                                        }

                                        //check
                                        if (strchr(gc_Command.cmdbuffer[gc_Command.bufindw], '*') != NULL)
                                        {
                                                byte checksum = 0;
                                                byte count = 0;
                                                while(gc_Command.cmdbuffer[gc_Command.bufindw][count] != '*') checksum
= checksum^gc_Command.cmdbuffer[gc_Command.bufindw][count++];
                                                //checksum
                                                strchr_pointer = strchr(gc_Command.cmdbuffer[gc_Command.bufindw], '*');
                                                if
((int)(strtod(&gc_Command.cmdbuffer[gc_Command.bufindw][strchr_pointer                -
```

```
gc_Command.cmdbuffer[gc_Command.bufindw] + 1], NULL)) != checksum)
                                {
                                        serial_count = 0; //clear buffer
                                        return;
                                }
                        }
                        else                                    //if no check , error and return
                        {
                                serial_count = 0;       //clear buffer
                                return;
                        }
                        gcode_LastN = gcode_N;
                }
                else        // if we don't receive 'N' but still see '*',error and return
                {
                        if ((strchr(gc_Command.cmdbuffer[gc_Command.bufindw], '*') != NULL))
                        {
                                serial_count = 0;       //clear buffer
                                return;
                        }
                }

                //one command over readed
                gc_Command.bufindw = (gc_Command.bufindw + 1)%BUFSIZE;
                gc_Command.buflen += 1;
        }
        serial_count = 0;       //clear buffer
    }
    else  //continue get char
    {
        if (!comment_mode)
            gc_Command.cmdbuffer[gc_Command.bufindw][serial_count++]  =  serial_char;//reading
command
    }
  }
}

//check current command contains character
bool code_seen(char code)
{
    strchr_pointer = strchr(gc_Command.cmdbuffer[gc_Command.bufindr], code);
    return(strchr_pointer != NULL);                 //Return True if a character was found
}

//get behind the keyword values
float code_value()
{
    return(strtod(&gc_Command.cmdbuffer[gc_Command.bufindr][strchr_pointer                          -
gc_Command.cmdbuffer[gc_Command.bufindr] + 1], NULL));
}

//get behind the keyword 'X','Y','Z','E','F' valuees,into destination[] and feedrate
void get_commandValue()
{
    int i=0;
    for (i=0; i < NUM_AXIS; i++)
    {
        if (code_seen(axis_codes[i]))
        {
            move_block.destination[i]                   =                   (float)code_value()                 +
sys_state.relative_mode*move_block.current_position[i];
        }
```

```
            else move_block.destination[i] = move_block.current_position[i];            //Are these else lines
really needed?
        }
        if (code_seen('F'))
        {
            if (code_value() > 0.0f) move_block.feedrate = code_value();
        }
}

void get_PCommandValue()
{
        if (code_seen('V'))
        {
            gc_Command.value_V = code_value();
        }
}
```