MJKZZ Serial Camera Motion Controller

Overview

MJKZZ Serial Camera Motion Controller is a versatile device that allows you to perform both motion control and camera triggering via serial bus. It can be applied in many camera motion applications, such as focus stacking, time lapse, astro-photography, etc.

This controller has a unique feature -- the torque of motor can be controlled digitally via command sent to it. It can supply up to 1.5A of current to each phase of bi-polar step motor. And power settings are saved in EEPROM so after power cycle, the same power setting will be restored.

Another feature of this controller is its ability to be used in a networked environment, such as zigbee network for wireless or RS485 network. This is essential for those application that has multiple camera/motors -- each will have its own address.

Hardware Features

This controller has the following ports:

- Power Port -- this port can accept center positive power plugs and voltage can be between 9V to 24V. Do not exceed 24V and 12V is the best choice.
- Camera Port -- this is where camera should be plugged in. **Control circuitry employs optical isolator where there is no direct connection between camera and this device**.
- Limit Port -- this port is a dual purpose port. It can be used as limit switch port where switch can be installed to automatically stop the rail when the switch is triggered. It can also be hooked up to a rotary switch so that the rail can be controlled by it. Note, this port can only be in one mode, either limit switch mode or rotary mode, not both at same time. Mode can be set via software command and will stay that way until it is changed, even after power off.
- Serial Port -- this is essentially a standard TTL serial port set at 9600 baud rate, 1 stop bit and no parity. A wireless serial device can be connected to this port, such as bluetooth, zigbee device. It can also be connected to a wired network, such as RS485 for multiple camera/motor control.
- Four Pin Bi-polar Motor Port -- this is where a bi-polar step motor can be connected. The pins are arranged as A+ A- B+ B-. This controller can supply up to

1.5A current to motor per phase. Current can be controlled and changed via command. With this feature, a wide range of motor can be used with this controller.

Software Command Set

This controller has a rich set of commands communicated via serial bus in form of 8 byte message. The following is the data structure of message (C++)

```
typedef struct tagRailMessage
{
    BYTE    ucADD;
    BYTE    ucCMD;
    BYTE    ucIDX;
    BYTE    ucMSG[4];
    BYTE    ucSUM;
} RailMessage;
```

- ucADD -- device address, factory default for each device is 0x01, it can be change via a message described later. It has following meanings:
  - When master device, such as PC or mobile phone, this field must be between 0x01 to 0x7F. When the addressed device responds, the high bit is set (ie, or'ed with 0x80). An addressed device MUST responds with an acknowledgment message back if the message is NOT a broadcast message.
  - When ucADD is set to 0xFF, the message is called a broadcast message, all devices on the bus should respond to it but NO device should respond to broadcast message
- ucCMD -- command byte. If the message is NOT a broadcast message (ie ucADD is not 0xFF), the device will set the high bit of this field to 1 (ie, or'ed with 0x80) if the message sent from a master is a valid one to acknowledge the sender (master) that the return message has meaningful data. Otherwise, the same CMD will be returned back to sender (master) that the message is an invalid command.
- ucIDX -- some commands can have multiple meanings, such as GREG or SREG to get or set register value. This is also for future command expansion.
- ucMSG[4] -- these four bytes are data fields for a command. The meaning is different for each command, please see below for each command. When these four bytes represent a long integer value, the highest order byte is stored in ucMSG[0] and lowest byte is stored in ucMSG[3]. Here is pseudo code:

```
smsg.ucMSG[0]   = (BYTE) (lwValue >> 24);
smsg.ucMSG[1]   = (BYTE) (lwValue >> 16);
smsg.ucMSG[2]   = (BYTE) (lwValue >> 8 );
```

```
smsg.ucMSG[3]   = (BYTE) (lwValue);
```

The following is pseudo code to unpack ucMSG to a long integer

```
lwValue    = rmsg.ucMSG[0];
lwValue    = lwValue << 8;
lwValue += rmsg.ucMSG[1];
lwValue    = lwValue << 8;
lwValue += rmsg.ucMSG[2];
lwValue    = lwValue << 8;
lwValue += rmsg.ucMSG[3];
```

- ucSUM -- this is the check sum byte, calculation is simple, sum of all BYTE value of previous 7 bytes without overflow bits.

Here is the list of commands

```
#define     CMD_GVER        'v'   // get version
#define     CMD_SCAM        'A'   // activate camera for lwValue
#define     CMD_SFCS        'a'   // activate half pressed
#define     CMD_MOVE        'M'   // move by positive or negative increment
#define     CMD_SREG        'R'   // set register value
#define     CMD_GREG        'r'   // get register value
#define     CMD_SPOS        'P'   // set position
#define     CMD_GPOS        'p'   // get position
#define     CMD_SSPD        'S'   // set speed
#define     CMD_GSPD        's'   // get speed
#define     CMD_SSET        'I'   // set settle time
#define     CMD_GSET        'i'   // get settle time
#define     CMD_SHLD        'D'   // set hold period
#define     CMD_GHLD        'd'   // get hold period
#define     CMD_SBCK        'K'   // set backlash
#define     CMD_GBCK        'k'   // get backlash
#define     CMD_SLAG        'G'   // set shutter lag
#define     CMD_GLAG        'g'   // get shutter lag
#define     CMD_SSPS        'B'   // set start position
#define     CMD_GSPS        'b'   // get start position
#define     CMD_SCFG        'F'   // set configuration flag
#define     CMD_GCFG        'f'   // get configuration flag
#define     CMD_SEPS        'E'   // set ending position
#define     CMD_GEPS        'e'   // get ending position
#define     CMD_SCNT        'N'   // set number of captures
#define     CMD_GCNT        'n'   // get number of captures
#define     CMD_SSSZ        'Z'   // set step size
```

```
#define     CMD_GSSZ        'z'   // get step size
#define     CMD_EXEC        'X'   // execute commands
#define     CMD_STOP        'x'   // stop execution of commands
```

Command Messages

Command CMD_GVER 'v'

Get version command, if the addressed device exists on the bus, it will respond with its version number.

Command CMD_SCAM 'A'

This commands tells the addressed device to activate, ie taking a picture, pull both focus and trigger signal of camera for number of milliseconds specified in ucMSG bytes as long integer. Essentially taking a picture.

Command CMD_SFCS 'a'

This is the command to activate half pressed, ie focus, signal of camera for number of milliseconds specified in ucMSG bytes as long integer. This is useful to tell the camera to focus but not to take a picture.

Command CMD_MOVE 'M'

This is to move the rail by positive or negative increment specified in ucMSG bytes as signed long integer. Positive value move the rail forward and negative value moves it backward. The ucIDX field can have two values as following:

```
enum      enumMOVE
{
      move_normal,
      move_bounded,

      move_last,
};
```

The value “move_normal” means move the rail without any boundary, this is used to move the rail freely and is useful to set limiting boundaries. For example, use this command to move the rail forward 10mm as maximum position, then the subsequent “move_bounded” command will be bounded by this value, ie, the rail will not move beyond 10mm with “move_bounded” command.

Command CMD_SREG 'R'

This is to set register value specified in ucIDX value. The following are some valid values for ucIDX (starting 100) :

```
enum       enumREG
{
      reg_STAT     =    100,
      reg_LPWR,
      reg_HPWR,
      reg_MSTEP,
      reg_MAXP,
      reg_EXEC,

      reg_ADDR,

      reg_last,
};
```

- reg_STAT -- this is for read register command as described below.
- reg_LPWR -- this is to set power level when motor is idling
- reg_HPWR -- this is to set power level when motor is in use.
- reg_MSTEP -- this is to set micro step mode for the step motor
- reg_MAXP -- this is to set maximum position value
- reg_EXEC -- read only for GREG command

Value for power setting range from 1 to 12 which corresponds to 0.125A to 1.5A in 0.125A increment. This value will be stored in ucMSG as long integer.

MSTEP values are as following:

```
#define MOTOR_4STEP              0
#define MOTOR_8STEP              1
#define MOTOR_HSTEP              2
```

For 1/4 micro step, 1/8 micro step, and 1/32 micro step.

Command CMD_GREG 'r'

This is to get register value specified in ucIDX value which is same enum as CMD_SREG.

- reg_STAT -- get status, the values returned are packed in ucMSG fields. ucMSG[0] is the version number, ucMSG[1] contains idle power setting, ucMSG[2] contains operating power setting and ucMSG[3] contains current micro step value.

- reg_EXEC -- returns current execution mode in ucMSG[0], and if in stacking mode, the nth step it is stacking, in combination of ucMSG[1] (high order byte) and ucMSG[2] (low order byte) as an integer. The execution mode ucMSG[0] indicate the device is currently in stacking mode if (ucMSG[0] & 0x03) is none zero.

Command CMD_SPOS 'P'

This is to set position of rail, if the rail is not at this position, it will be moved to this position. The position value is contained in ucMSG bytes as long integer.

Command CMD_GPOS 'p'

This is to get current position of rail. The returned value is stored in ucMSG in reply message as long integer.

Command CMD_SSPD 'S'

This is to set speed of rail. Beside setting micro steps, this controller allow speed to be set. The speed value can be specified in ucMSG as long integer, though range of value is 0 to 255. However, it is recommended to use range of 0 to 3 because if the value is too large, the motor will be running too slow. The larger the value, the slower it is.

Command CMD_GSPD 's'

This is to get current speed, the returned value is in ucMSG as long integer

Command CMD_SSET 'I'

This is to set settle time in milliseconds where its value is stored in ucMSG bytes as long integer. Settle time is the duration device delays after moving and before triggering camera. This is used to reduce mechanical vibration until everything are settled.

Command CMD_GSET 'i'

This is to get current settle time and its value is stored in ucMSG bytes as long integer

Command CMD_SHLD 'D'

This is the time duration the device must wait after triggering the camera before moving the rail. This is the time duration that should be longer than actual camera shutter open duration. For example, if camera is set to have 1 second exposure time, this parameter must be longer than 1 second before it moves because else the image

will be blurry. In another case, if camera is set to have a timer after mirror is locked up, then this value must be longer than the timer duration. In any case, this parameter should be set to be longer than the duration camera takes an images.

Command CMD_GHLD 'd'

This is to get current hold period as described above and its value is stored in usMSG bytes as long integer.

Command CMD_SBCK 'K'

This is to set backlash for a particular rail system. Backlash refers to mechanical gap when motor/rail reverses its moving direction. If not handled, moving a rail forward certain distance and move it back the same distance will not result in the same position as it started. Note this value must be or translated to the value when motor is set to 1/32 micro step. For example, if a rail system has a pitch value of 2mm and is using 200 step per turn motor, at 1/32 micro stepping, each step is 0.3125um, to compensate a backlash of 10um, the value must be 10/0.3125 = 32. The backlash compensation value should be stored in ucMSG bytes as long integer

Command CMD_GBCK 'k'

This is to get backlash value in the current system and it is returned in ucMSG bytes as long integer.

Command CMD_SLAG 'G'

This is to set shutter lag. Shutter lag is the minimum duration a camera will recognize a picture taking signal when both focus and trigger signals are set. This value can also be used for shutter open duration for Bulb mode operation of a camera. For most cameras, 200ms duration is enough, but for some, 250ms is needed to successful y trigger it. The value is packed in ucMSG bytes as long integer in milliseconds.

Command CMD_GLAG 'g'

This is to get current shutter lag parameter and its value is stored in ucMSG bytes as long integer.

Command CMD_SSPS 'B'

This is to set start position for the rail. It value is packed in ucMSG bytes as long integer. For any motion and click operation, a starting position must be specified, as well as an ending position. Important note, when CMD_EXEC is issued, the rail will be moved to this starting position automatically.

Command CMD_GSPS 'b'

This is to get current start position stored in the device. The value is packed in the ucMSG bytes of the returned message.

Command CMD_SEPS 'E'

This is to set ending position for a motion controlled camera operation. The value is packed in ucMSG as long integer.

Command CMD_GEPS 'e'

This is to get current ending position and its value is packed in ucMSG as long integer.

Command CMD_SCFG 'F'

This is to set configuration flag. Currently, there is only one configuration flag -- setting a flag to change Limit port to Rotary switch port. The flag is one byte long and its value is stored in ucMSG[0]. By default, the Limit port is actually set to Rotary Switch port and a rotary switch can be attached to move the rail forward or back. However, for those system that needs a limit switch, the Limit port can be changed to limit switch port where a set of switch can be used to stop the rail automatically when it reaches either ends.

Command CMD_GCFG 'f'

This is to get configuration flag as described above, the returned value is stored in ucMSG[0] of the returned message.

Command CMD_SCNT 'N'

This is to set number of captures or images. Its value is stored in ucMSG bytes as long integer. Note, it is also necessary to specify a step size even though it can be determined. The reason is that, this way, it is possible not to reach end position if desired by specifying shorter step size or less number of captures.

Command CMD_GCNT 'n'

This is to get current number of captures, its value is packed in ucMSG bytes as long integer.

Command CMD_SSSZ 'Z'

This is to set step size. Step size can be calculated from starting and ending position, but also can be set to other values to achieve certain effects. Its value is packed in ucMSG bytes as long integer

Command CMD_GSSZ 'z'

This is to get current step size as described above and its value is packed in ucMSG bytes as long integer.

Command CMD_EXEC 'X'

This is the command that starts execution of motion and camera operation. Progress can be obtained by querying CMD_GREG with reg_EXEC as ucIDX.

Command CMD_STOP 'x'

This command stops ALL actions and returns current position the rail is stopped at in its ucMSG bytes as long integer. It can also be used to stop a moving rail, for example when an CMD_MOVE command is in effect. It is also an convenient and "safe" way to query current rail position.

Frequent Asked Questions

Q: How do I find out all devices in a bused environment.
A: You can use CMD_GVER message starting from address 0x01 to 0x7F (or whatever the end address is), if there is a device with the address sent, it will respond with return messages. If there is no such device with that address, it will timeout and after number of retries, we can safely determine that there is no such device.

Q: I am worried about connecting this device to my camera.
A: This device uses optical isolation to separate its circuitry from that of camera, so it is safe.

Q: Can I hook up another step motor?
A: Yes, this device allows you to control current supplied to step motor, up to 1.5A per phase.

Q: Can I use my iPhone or iPad or an Android phone to control it?
A: Yes, as long as the device support Bluetooth BLE and you have our Bluetooth adapter installed. For iOS device, you need iPhone 4s and up, or iPad 3 and up. For Android phones, you will need Android 4.3 phones equipped with Bluetooth BLE (Bluetooth 4.0). The iOS app can be downloaded from Apple app store and the Android version can be downloaded from our website support section.