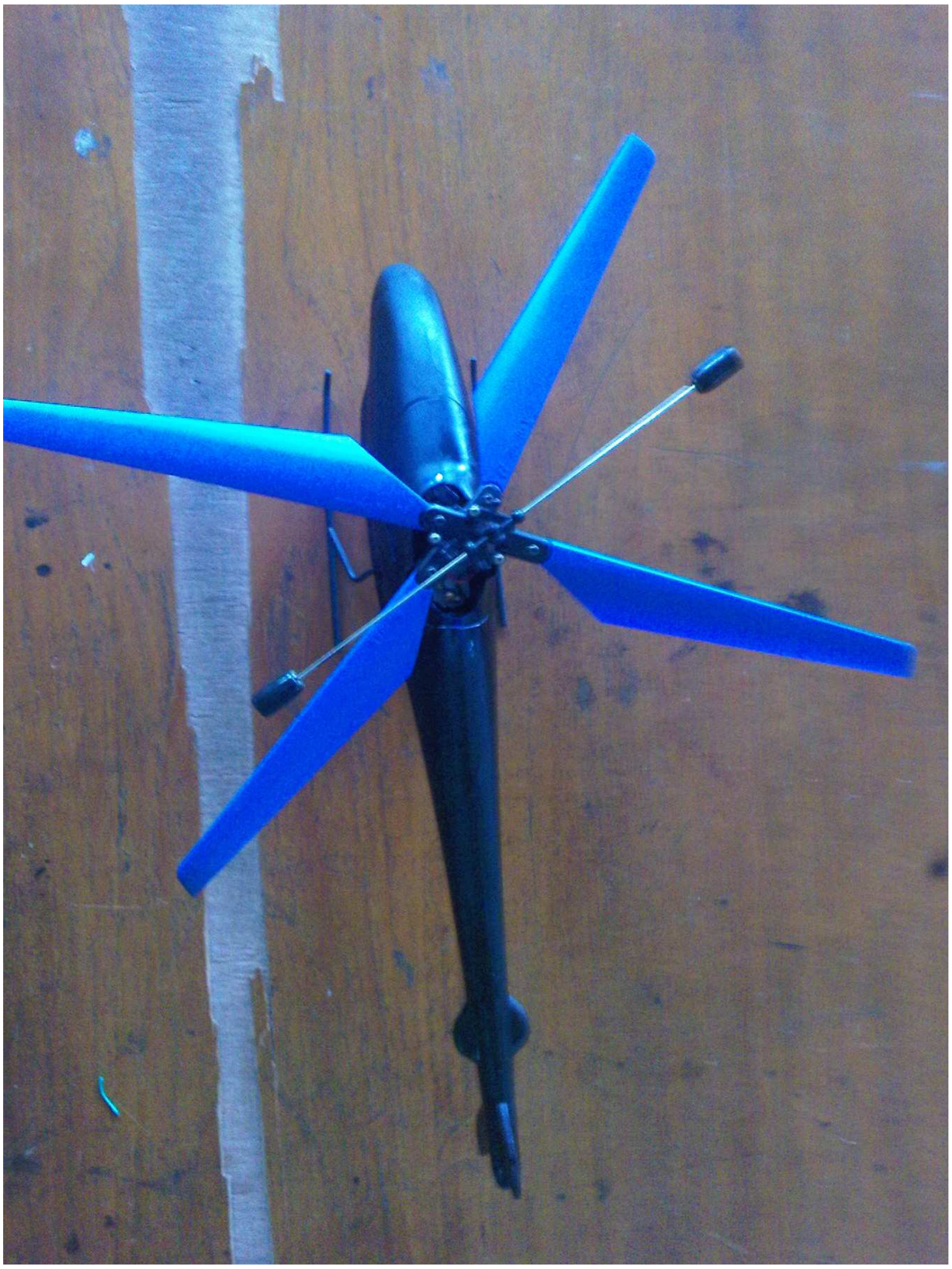
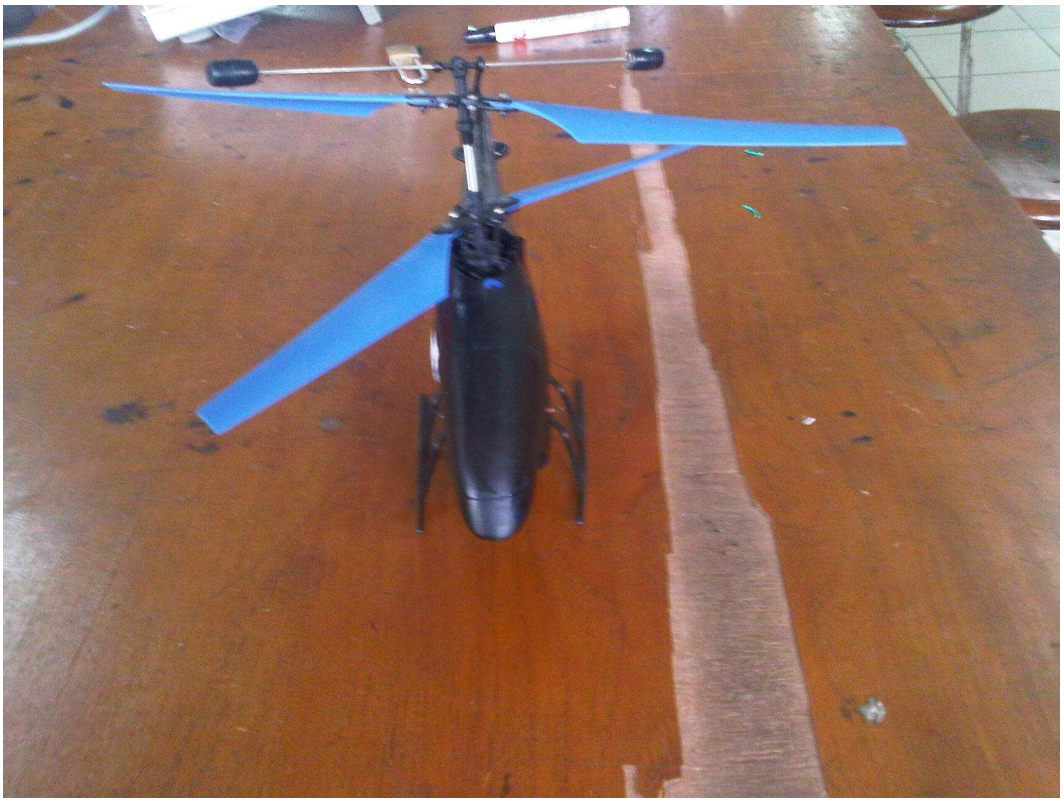
LAMPIRAN A

FOTO *Radio Control* Helikopter dan Pengendalinya

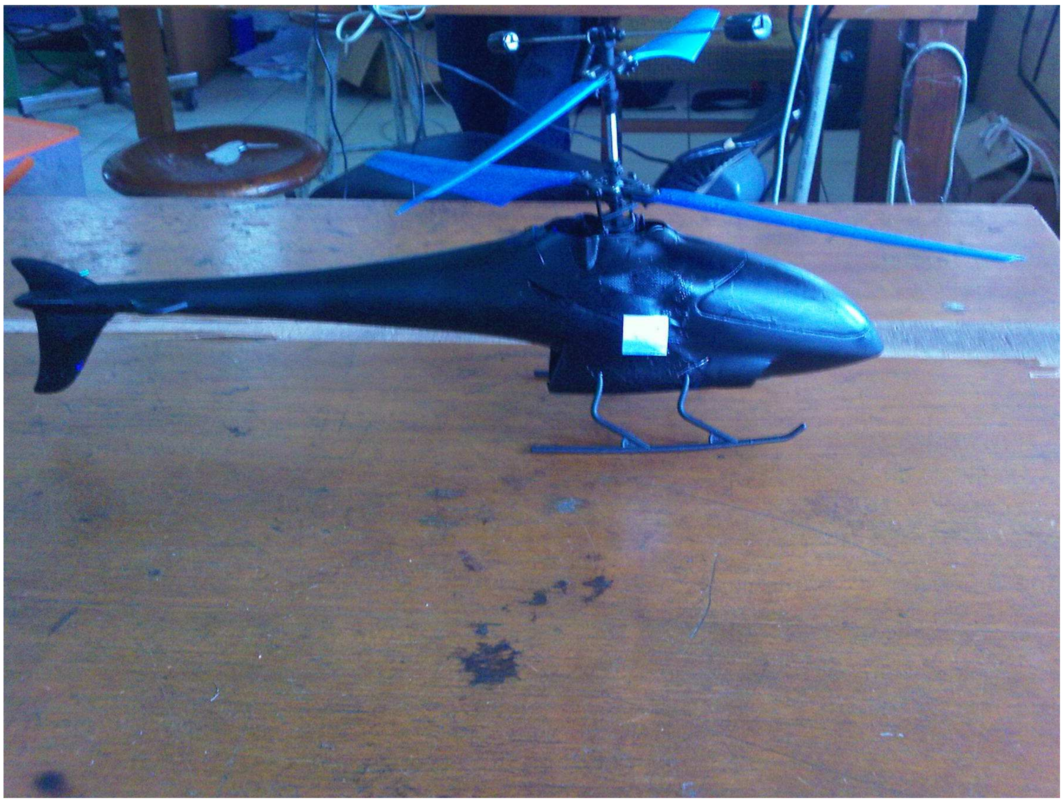
Tampak Atas



Tampak Depan

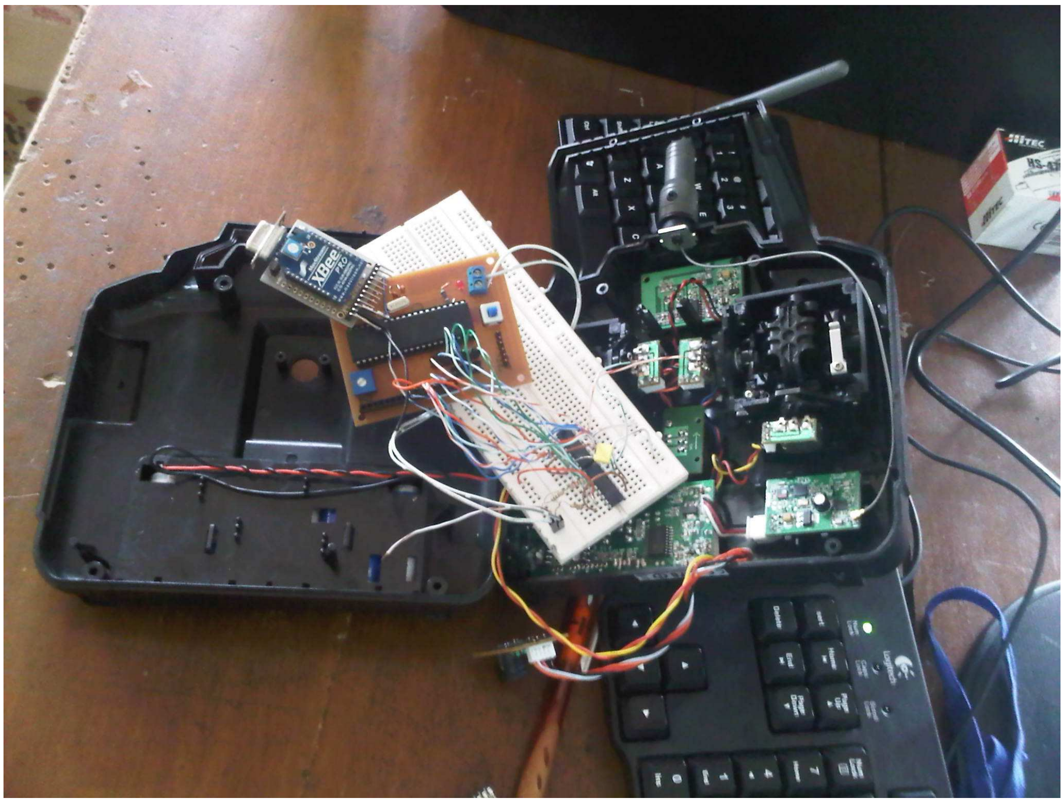


Tampak Samping



Tampak Belakang

Pengendali



LAMPIRAN B

PROGRAM PADA *MICROSOFT VISUAL BASIC 6* DAN PENGONTROL MIKRO ATMEGA16

MICROSOFT VISUAL BASIC 6

```
//---------------------------------------------------------------------------------------------
//                                        Tombol  Start
//---------------------------------------------------------------------------------------------
Private Sub Command1_Click()
   Label4.Caption = "0"
   Timer5.Enabled = True
   MSComm1.Output = "TC6 64 61 115 94 196" & vbCrLf
   If Option1.Value = True Then
      Timer1.Enabled = True
   End If
   If Option2.Value = True Then
      Timer2.Enabled = True
   End If
   If Option3.Value = True Then
      Timer3.Enabled = True
   End If
   If CStr(Label6.Caption) > 0 Then
      Timer4.Enabled = True
   End If
End Sub
//---------------------------------------------------------------------------------------------
//                                        Tombol  Stop
//---------------------------------------------------------------------------------------------
Private Sub Command2_Click()
      Timer1.Enabled = False
      Timer2.Enabled = False
      Timer3.Enabled = False
      Timer4.Enabled = False
      Timer5.Enabled = False
      For i = 1 To 200
```

```
        MSComm1.Output = "  " & vbCrLf
        Next i
        RichTextBox1.Text = " "
End Sub
//-----------------------------------------------------------------------------------------------
//                                          Program Awal
//-----------------------------------------------------------------------------------------------
Private Sub Form_Load()
    MSComm1.PortOpen = True
    MSComm1.PortOpen = True
    Timer1.Enabled = False
    Timer2.Enabled = False
    Timer3.Enabled = False
    Timer4.Enabled = False
    Timer5.Enabled = False
    Option1.Value = True
End Sub
//-----------------------------------------------------------------------------------------------
//                                       Program pada Timer1
//-----------------------------------------------------------------------------------------------
Private Sub Timer1_Timer()
    Do While x < X1
    x = Value(Label6.Caption)
    y = Value(Label7.Caption)
    If x < X1 Then
        g1 = 1
        Else: g1 = 0
        End If
    If y < Y1 Then
        g2 = 2
        Else: g2 = 0
        End If
    If y > Y1 Then
        g3 = 4
        Else: g3 = 0
        End If
    g = g1 + g2 + g3
    MSComm2.Output = g
    Loop
    Do While x < X2
    x = Value(Label6.Caption)
    y = Value(Label7.Caption)
    If x < X2 Then
        g1 = 1
        Else: g1 = 0
        End If
    If y < Y2 Then
        g2 = 2
        Else: g2 = 0
        End If
    If y > Y2 Then
        g3 = 4
        Else: g3 = 0
        End If
    g = g1 + g2 + g3
    MSComm2.Output = g
    Loop
    Do While x < X3
    x = Value(Label6.Caption)
    y = Value(Label7.Caption)
    If x < X3 Then
        g1 = 1
        Else: g1 = 0
        End If
    If y < Y3 Then
        g2 = 2
        Else: g2 = 0
        End If
    If y > Y3 Then
        g3 = 4
        Else: g3 = 0
        End If
    g = g1 + g2 + g3
    MSComm2.Output = g
    Loop
    Do While x < X4
    x = Value(Label6.Caption)
    y = Value(Label7.Caption)
    If x < X4 Then
        g1 = 1
        Else: g1 = 0
        End If
    If y < Y4 Then
        g2 = 2
        Else: g2 = 0
        End If
    If y > Y4 Then
        g3 = 4
        Else: g3 = 0
        End If
    g = g1 + g2 + g3
    MSComm2.Output = g
    Loop
```

```
   Do While x < x5
   x = Value(Label6.Caption)
   y = Value(Label7.Caption)
   If x < x5 Then
      g1 = 1
      Else: g1 = 0
      End If
   If y < Y5 Then
      g2 = 2
      Else: g2 = 0
      End If
   If y > Y5 Then
      g3 = 4
      Else: g3 = 0
      End If
   g = g1 + g2 + g3
   MSComm2.Output = g
   Loop
   If x > x5 Then
   Timer1.Enabled = False
   Timer4.Enabled = False
End Sub
//--------------------------------------------------------------------------------------------------
//                                        Program pada Timer2
//--------------------------------------------------------------------------------------------------
Private Sub Timer2_Timer()
   Do While x < X1
   x = Value(Label6.Caption)
   y = Value(Label7.Caption)
   If x < X1 Then
      g1 = 1
   Else: g1 = 0
   End If
If y < Y1 Then
   g2 = 2
   Else: g2 = 0
   End If
If y > Y1 Then
   g3 = 4
   Else: g3 = 0
   End If
g = g1 + g2 + g3
MSComm2.Output = g
Loop
Do While x < X2
x = Value(Label6.Caption)
y = Value(Label7.Caption)
If x < X2 Then
   g1 = 1
   Else: g1 = 0
   End If
If y < Y2 Then
   g2 = 2
   Else: g2 = 0
   End If
If y > Y2 Then
   g3 = 4
   Else: g3 = 0
   End If
g = g1 + g2 + g3
MSComm2.Output = g
Loop
```

```
Do While x < X3
x = Value(Label6.Caption)
y = Value(Label7.Caption)
If x < X3 Then
    g1 = 1
    Else: g1 = 0
    End If
If y < Y3 Then
    g2 = 2
    Else: g2 = 0
    End If
If y > Y3 Then
    g3 = 4
    Else: g3 = 0
    End If
g = g1 + g2 + g3
MSComm2.Output = g
Loop
Do While x < X4
x = Value(Label6.Caption)
y = Value(Label7.Caption)
If x < X4 Then
    g1 = 1
    Else: g1 = 0
    End If
If y < Y4 Then
    g2 = 2
    Else: g2 = 0
    End If
If y > Y4 Then
    g3 = 4
```

```
        Else: g3 = 0
        End If
    g = g1 + g2 + g3
    MSComm2.Output = g
    Loop
    Do While x < x5
    x = Value(Label6.Caption)
    y = Value(Label7.Caption)
    If x < x5 Then
        g1 = 1
        Else: g1 = 0
        End If
    If y < Y5 Then
        g2 = 2
        Else: g2 = 0
        End If
    If y > Y5 Then
        g3 = 4
        Else: g3 = 0
        End If
    g = g1 + g2 + g3
    MSComm2.Output = g
    Loop
    If x > x5 Then
    Timer2.Enabled = False
    Timer4.Enabled = False
End Sub
//---------------------------------------------------------------------------------------------
//                                  Program pada Timer3
//---------------------------------------------------------------------------------------------
Private Sub Timer3_Timer()
```

```
Do While x < X1
x = Value(Label6.Caption)
y = Value(Label7.Caption)
If x < X1 Then
   g1 = 1
   Else: g1 = 0
   End If
If y < Y1 Then
   g2 = 2
   Else: g2 = 0
   End If
If y > Y1 Then
   g3 = 4
   Else: g3 = 0
   End If
g = g1 + g2 + g3
MSComm2.Output = g
Loop
Do While x < X2
x = Value(Label6.Caption)
y = Value(Label7.Caption)
If x < X2 Then
   g1 = 1
   Else: g1 = 0
   End If
If y < Y2 Then
   g2 = 2
   Else: g2 = 0
   End If
If y > Y2 Then
   g3 = 4
```

```
    Else: g3 = 0
    End If
g = g1 + g2 + g3
MSComm2.Output = g
Loop
Do While x < X3
x = Value(Label6.Caption)
y = Value(Label7.Caption)
If x < X3 Then
    g1 = 1
    Else: g1 = 0
    End If
If y < Y3 Then
    g2 = 2
    Else: g2 = 0
    End If
If y > Y3 Then
    g3 = 4
    Else: g3 = 0
    End If
g = g1 + g2 + g3
MSComm2.Output = g
Loop
Do While x < X4
x = Value(Label6.Caption)
y = Value(Label7.Caption)
If x < X4 Then
    g1 = 1
    Else: g1 = 0
    End If
If y < Y4 Then
```

```
    g2 = 2
    Else: g2 = 0
    End If
If y > Y4 Then
    g3 = 4
    Else: g3 = 0
    End If
g = g1 + g2 + g3
MSComm2.Output = g
Loop
Do While x < x5
x = Value(Label6.Caption)
y = Value(Label7.Caption)
If x < x5 Then
    g1 = 1
    Else: g1 = 0
    End If
If y < Y5 Then
    g2 = 2
    Else: g2 = 0
    End If
If y > Y5 Then
    g3 = 4
    Else: g3 = 0
    End If
g = g1 + g2 + g3
MSComm2.Output = g
Loop
If x > x5 Then
Timer3.Enabled = False
Timer4.Enabled = False
```

```
End Sub
//---------------------------------------------------------------------------------------------
//                      Program pada Timer4 (Penghitung Lamanya Proses)
//---------------------------------------------------------------------------------------------
Private Sub Timer4_Timer()
   Label4.Caption = Val(Label4.Caption) + 1
End Sub
//---------------------------------------------------------------------------------------------
//                       Program pada Timer5 (Pengambil Data CMUCam2+)
//---------------------------------------------------------------------------------------------
Private Sub Timer5_Timer()
   Dim Data As Variant
   Data = MSComm1.Input
   RichTextBox1.Text = Data
   Data = Split(Data, " ")
   Label6.Caption = Data(1)
   Label7.Caption = Data(2)
End Sub
```

PENGONTROL MIKRO ATMEGA16

```
/*****************************************************
This program was produced by the
CodeWizardAVR V1.25.3 Professional
Automatic Program Generator
© Copyright 1998-2007 Pavel Haiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com

Project : Pengontrol Remote RC Heli
Version : 1
Date    : 9/27/2010
Author  : IVan/PiNgu/APui/ToPui
Company : UKM
Comments:


Chip type           : ATmega16
Program type        : Application
Clock frequency     : 11.059200 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 256
*****************************************************/

#include <mega16.h>
#include <delay.h>
#define RXB8 1
#define TXB8 0
#define UPE 2
#define OVR 3
#define FE 4
```

```
#define UDRE 5
#define RXC 7

#define FRAMING_ERROR (1<<FE)
#define PARITY_ERROR (1<<UPE)
#define DATA_OVERRUN (1<<OVR)
#define DATA_REGISTER_EMPTY (1<<UDRE)
#define RX_COMPLETE (1<<RXC)

// USART Receiver buffer
#define RX_BUFFER_SIZE 8
char rx_buffer[RX_BUFFER_SIZE];

#if RX_BUFFER_SIZE<256
unsigned char rx_wr_index,rx_rd_index,rx_counter;
#else
unsigned int rx_wr_index,rx_rd_index,rx_counter;
#endif

// This flag is set on USART Receiver buffer overflow
bit rx_buffer_overflow;

// USART Receiver interrupt service routine
interrupt [USART_RXC] void usart_rx_isr(void)
{
char status,data;
status=UCSRA;
data=UDR;
if ((status & (FRAMING_ERROR | PARITY_ERROR | DATA_OVERRUN))==0)
   {
```

```
  rx_buffer[rx_wr_index]=data;
  if (++rx_wr_index == RX_BUFFER_SIZE) rx_wr_index=0;
  if (++rx_counter == RX_BUFFER_SIZE)
    {
    rx_counter=0;
    rx_buffer_overflow=1;
    };
  };
}

#ifndef _DEBUG_TERMINAL_IO_
// Get a character from the USART Receiver buffer
#define _ALTERNATE_GETCHAR_
#pragma used+
char getchar(void)
{
char data;
while (rx_counter==0);
data=rx_buffer[rx_rd_index];
if (++rx_rd_index == RX_BUFFER_SIZE) rx_rd_index=0;
#asm("cli")
--rx_counter;
#asm("sei")
return data;
}
#pragma used-
#endif

// Standard Input/Output functions
#include <stdio.h>
```

```
// Declare your global variables here
int G = 0;
void main(void)
{
// Declare your local variables here

// Input/Output Ports initialization
// Port A initialization
// Func7=Out Func6=Out Func5=Out Func4=Out Func3=Out Func2=Out
Func1=Out Func0=Out
// State7=0 State6=0 State5=0 State4=0 State3=0 State2=0 State1=0 State0=0
PORTA=0x00;
DDRA=0xFF;

// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In
Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTB=0x00;
DDRB=0x00;

// Port C initialization
// Func7=Out Func6=Out Func5=Out Func4=Out Func3=Out Func2=Out
Func1=Out Func0=Out
// State7=0 State6=0 State5=0 State4=0 State3=0 State2=0 State1=0 State0=0
PORTC=0x00;
DDRC=0xFF;

// Port D initialization
// Func7=Out Func6=Out Func5=In Func4=In Func3=In Func2=In Func1=In
Func0=In
```

```
// State7=0 State6=0 State5=T State4=T State3=T State2=T State1=T State0=T
PORTD=0x00;
DDRD=0xC0;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer 1 Stopped
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
```

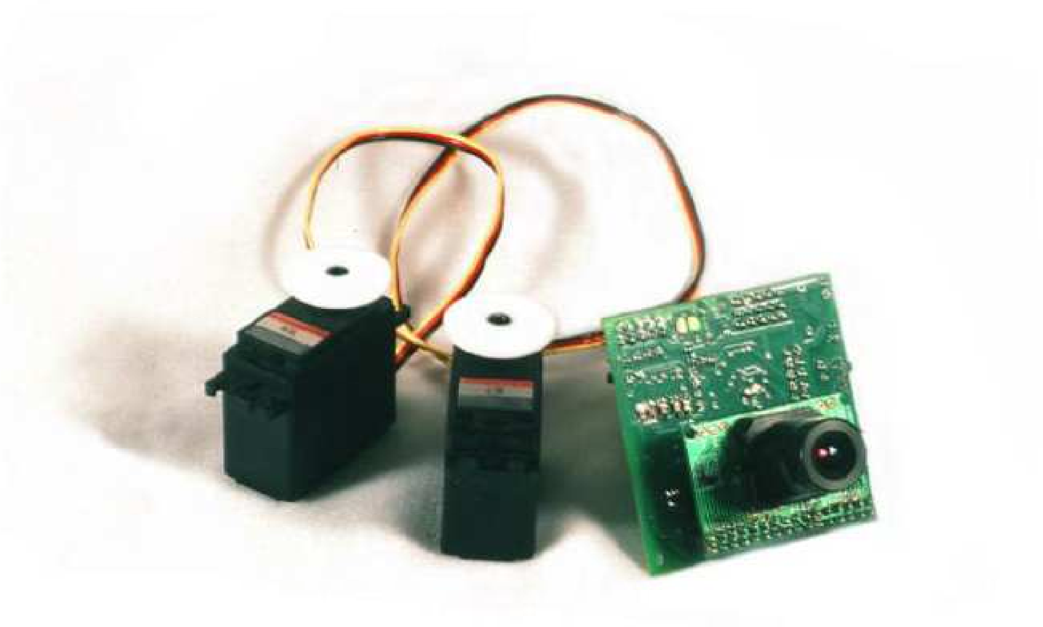
```
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// INT2: Off
MCUCR=0x00;
MCUCSR=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x00;

// USART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART Receiver: On
// USART Transmitter: Off
// USART Mode: Asynchronous
// USART Baud rate: 9600
```

```
UCSRA=0x00;
UCSRB=0x90;
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0x47;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;

// Global enable interrupts
#asm("sei")
//---------------------------------------------------------------------------------------------
//Program Untuk Memberikan Delay 7 Detik dan Kecepatan Motor DC dan Posisi
//Motor Servo yang Dimulai Setelah Batere Dipasangkan
//---------------------------------------------------------------------------------------------
delay_ms(7000);
PORTA=0x7F;
PORTC=0X2B;
while (1)
     {
     // Place your code here
//---------------------------------------------------------------------------------------------
//Program Untuk Mengendalikan Kecepatan Motor DC dan Posisi Motor Servo
//Berdasarkan “G”
//---------------------------------------------------------------------------------------------
      G=getchar();
      while (G==1)
       {
        PORTA=PORTA--;
       }
      while (G == 2)
       {
        PORTC--;
        delay_ms(300);
        PORTC++;
       }
      while (G == 3)
       {
        PORTA=PORTA--;
        PORTC=PORTC--;
        delay_ms(300);
        PORTC=PORTC++;
       }
      while (G == 4)
       {
        PORTA=PORTA++;
       }
      while (G == 5)
       {
        PORTA=PORTA++;
        PORTC=PORTC--;
        delay_ms(300);
        PORTC=PORTC++;
       }
     };
}
```

LAMPIRAN C

DATASHEET

CMUcam2 Vision Sensor

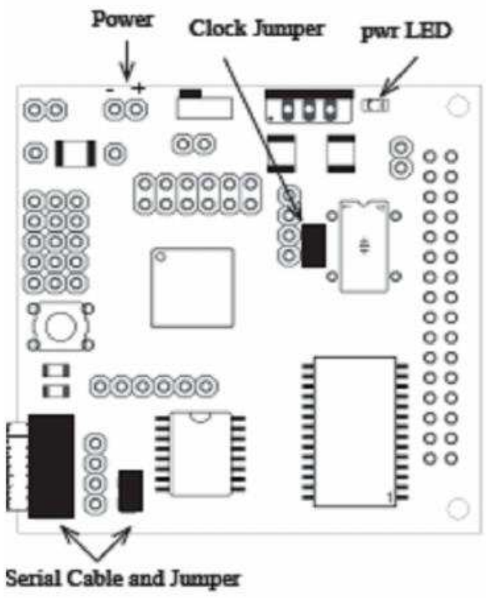


User Guide

Getting Started

Setting Up the Hardware

In order to initially test your CMUcam2, you will need a serial cable, a power adapter and a computer. The CMUcam2 can use a power supply which produces anywhere from 6 to 15 volts of DC power capable of supplying at least 200mA of current. This can be provided by either an AC adapter (possibly included) or a battery supply. These should be available at any local electronics store. The serial cable should have been provided with your CMUcam2. Make sure that you have the CMOS sensor board connected to the CMUcam2 board so that it is in the same orientation as the picture shows on the cover of this manual.



First, connect the power. Make sure that the positive side of your power plug is facing away from the main components on the board. If the camera came with an AC adapter, make sure that the connector locks into the socket correctly.

Now that the camera has power, connect the serial link between the camera and your computer. This link is required initially so that you can test and focus your camera. The serial cable should be connected so that the ribbon part of the cable faces away from the board. You must also connect the serial pass through jumper.

Check to make sure that the clock jumper is connected. This allows the clock to actively drive the processor.

Once everything is wired up, try turning the board on. The power LED should illuminate green and only one LED should remain on. Both LEDs turn on upon startup, and one turns off after the camera has been sucessfully configured.

Testing the Firmware

Once you have set the board up and downloaded the firmware, a good way to test the system is to connect it to the serial port of a computer.

Step 1: If one does not already exist, build a serial and/or power cable

Step 2: Plug both of them in.

Step 3: Open the terminal emulator of your choice.

Step 4: Inside the terminal emulator set the communication protocol to 115,200 Baud, 8 Data bits, 1 Stop bit, no parity, local echo on, no flow control and if possible turn on "add line feed" (add \n to a received \r). These setting should usually appear under "serial port" or some other similar menu option.

Step 5: Turn on the CMUcam2 board; the Power LED should light up and only one of the two status LEDs should remain on.

Step 6: You should see the following on your terminal emulator:

```
CMUcam2 v1.0 c6
:
```

If you have seen this, the board was able to successfully configure the camera and start the firmware.

Step 7: Type gv followed by the enter key. You should see the following:

```
:gv
ACK
CMUcam2 v1.0 c6
:
```

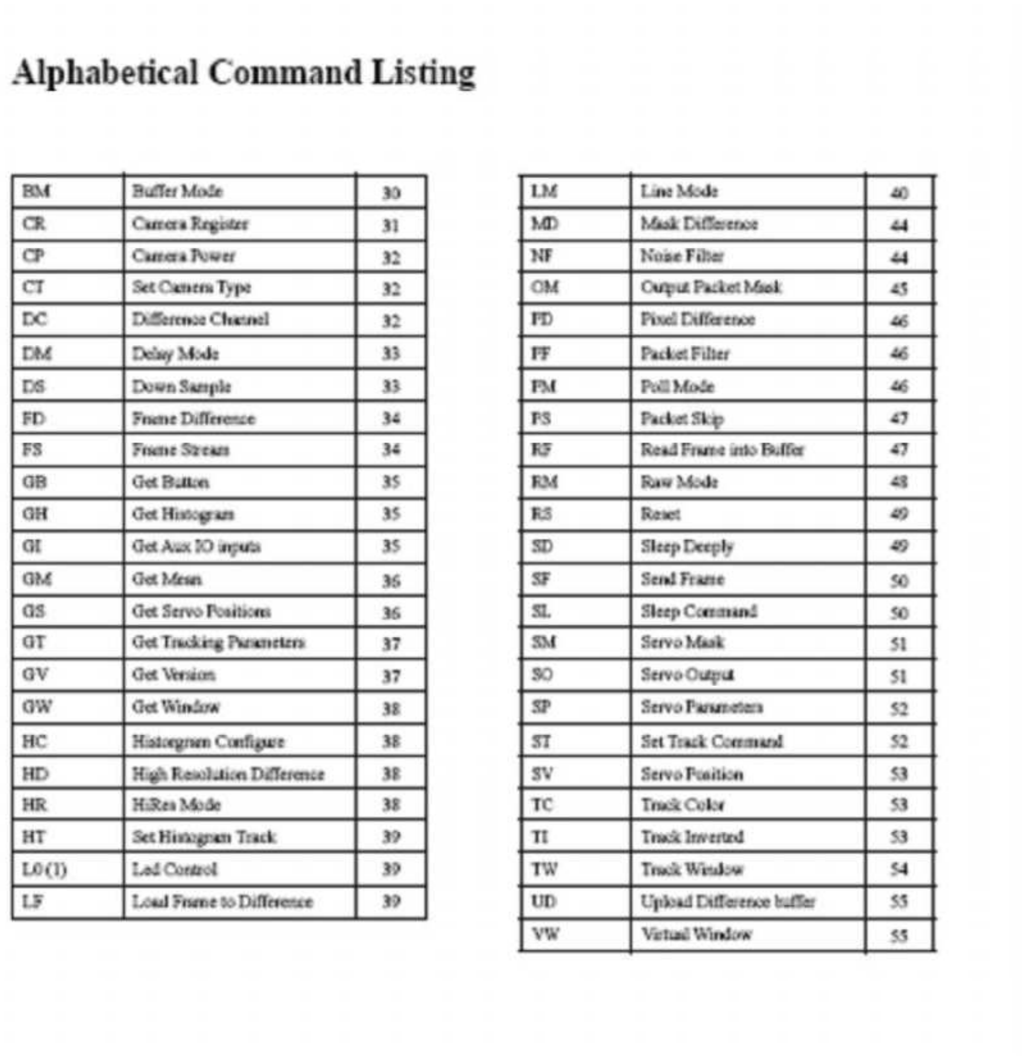
This shows the current version of the firmware. If this is successful, your computer's serial port is also configured correctly and both transmit and receive are working.

Serial Commands

The serial communication parameters are as follows:

- 1,200 to 115,200 Baud
- 8 Data bits
- 1 Stop bit
- No Parity
- No Flow Control (Not Xon/Xoff or Hardware)

All commands are sent using visible ASCII characters (123 is 3 bytes "123"). Upon a successful transmission of a command, the ACK string should be returned by the system. If there was a problem in the syntax of the transmission, or if a detectable transfer error occurred, a NCK string is returned. After either an ACK or a NCK, a \r is returned. When a prompt ('\r' followed by a ':') is returned, it means that the camera is waiting for another command in the idle state. White spaces do matter and are used to separate argument parameters. The \r (ASCII 13 carriage return) is used to end each line and activate each-command. If visible character transmission causes too much overhead, it is possible to use varying degrees of raw data transfer.



Alphabetical Command Listing

BM	Buffer Mode	30
CR	Camera Register	31
CP	Camera Power	32
CT	Set Camera Type	32
DC	Difference Channel	32
DM	Delay Mode	33
DS	Down Sample	33
FD	Frame Difference	34
FS	Frame Stream	34
GB	Get Button	35
GH	Get Histogram	35
GI	Get Aux IO inputs	35
GM	Get Mean	36
GS	Get Servo Positions	36
GT	Get Tracking Parameters	37
GV	Get Version	37
GW	Get Window	38
HC	Historgram Configure	38
HD	High Resolution Difference	38
HR	HiRes Mode	38
HT	Set Histogram Track	39
L0 (1)	Led Control	39
LF	Load Frame to Difference	39

LM	Line Mode	40
MD	Mask Difference	44
NF	Noise Filter	44
OM	Output Packet Mask	45
PD	Pixel Difference	46
PF	Packet Filter	46
PM	Poll Mode	46
PS	Packet Skip	47
RF	Read Frame into Buffer	47
RM	Raw Mode	48
RS	Reset	49
SD	Sleep Deeply	49
SF	Send Frame	50
SL	Sleep Command	50
SM	Servo Mask	51
SO	Servo Output	51
SP	Servo Parameters	52
ST	Set Track Command	52
SV	Servo Position	53
TC	Track Color	53
TI	Track Inverted	53
TW	Track Window	54
UD	Upload Difference buffer	55
VW	Virtual Window	55

\r

This command is used to set the camera board into an idle state. Like all other commands, you should receive the acknowledgment string "ACK" or the not acknowledge string "NCK" on failure. After acknowledging the idle command the camera board waits for further commands, which is shown by the ':' prompt. While in this idle state a \r by itself will return an "ACK" followed by \r and : character prompt. This is how you stop the camera while in streaming mode.

Example of how to check if the camera is alive while in the idle state:

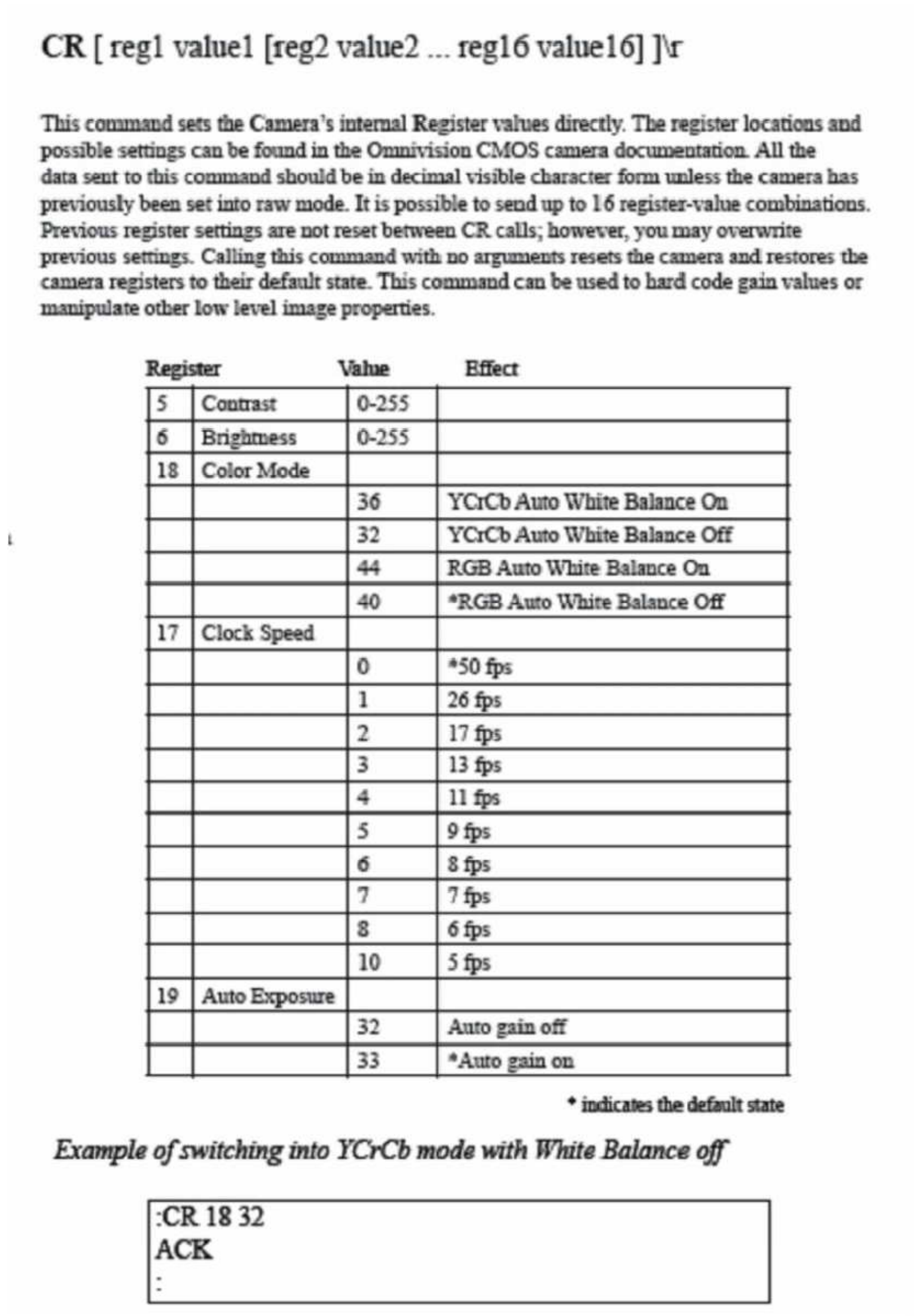
```
:
ACK
:
```

BM active \r

This command sets the mode of the CMUcam's frame buffer. A value of 0 (default) means that new frames are constantly being pushed into the frame buffer. A value of 1, means that only a single frame remains in the frame buffer. This allows multiple processing calls to be applied to the same frame. Instead of grabbing a new frame, all commands are applied to the current frame in memory. So you could get a histogram on all three channels of the same image and then track a color or call get mean and have these process a single buffered frame. Calling **RF** will then read a new frame into the buffer from the camera. When **BM** is off, **RF** is not required to get new frames.

Example of how to track multiple colors using buffer mode:

```
:BM 1
ACK
:PM 1
ACK
:TC 200 240 0 30 0 30
ACK
T 20 40 10 30 30 50 20 30
:RF
ACK
:TC 0 30 200 240 0 30
ACK
T 30 50 20 40 40 60 22 31
```



CR [reg1 value1 [reg2 value2 ... reg16 value16]]\r

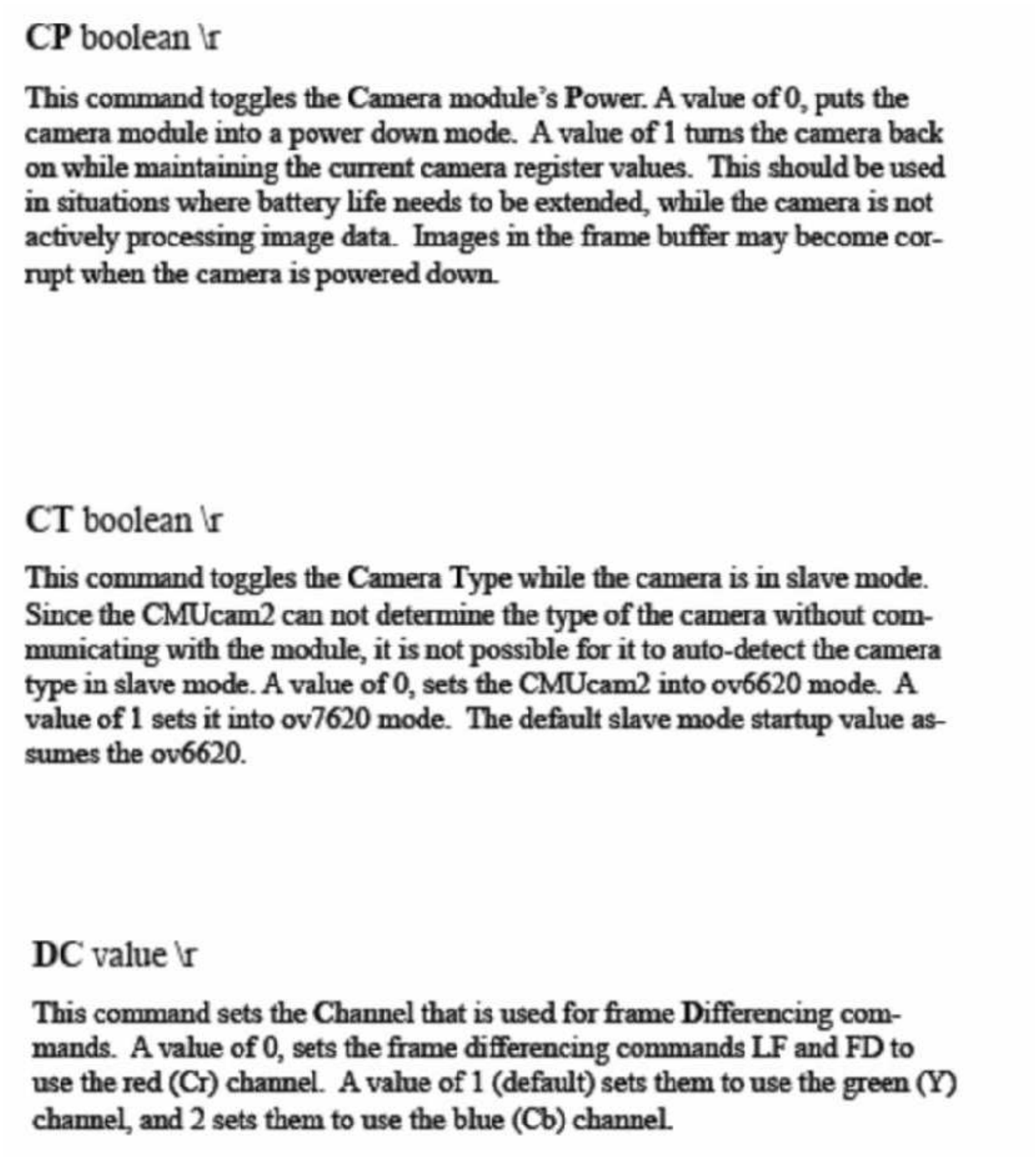
This command sets the Camera's internal Register values directly. The register locations and possible settings can be found in the Omnivision CMOS camera documentation. All the data sent to this command should be in decimal visible character form unless the camera has previously been set into raw mode. It is possible to send up to 16 register-value combinations. Previous register settings are not reset between CR calls; however, you may overwrite previous settings. Calling this command with no arguments resets the camera and restores the camera registers to their default state. This command can be used to hard code gain values or manipulate other low level image properties.

Register		Value	Effect
5	Contrast	0-255	
6	Brightness	0-255	
18	Color Mode		
		36	YCrCb Auto White Balance On
		32	YCrCb Auto White Balance Off
		44	RGB Auto White Balance On
		40	*RGB Auto White Balance Off
17	Clock Speed		
		0	*50 fps
		1	26 fps
		2	17 fps
		3	13 fps
		4	11 fps
		5	9 fps
		6	8 fps
		7	7 fps
		8	6 fps
		10	5 fps
19	Auto Exposure		
		32	Auto gain off
		33	*Auto gain on

* indicates the default state

Example of switching into YCrCb mode with White Balance off

```
:CR 18 32
ACK
:
```



CP boolean \r

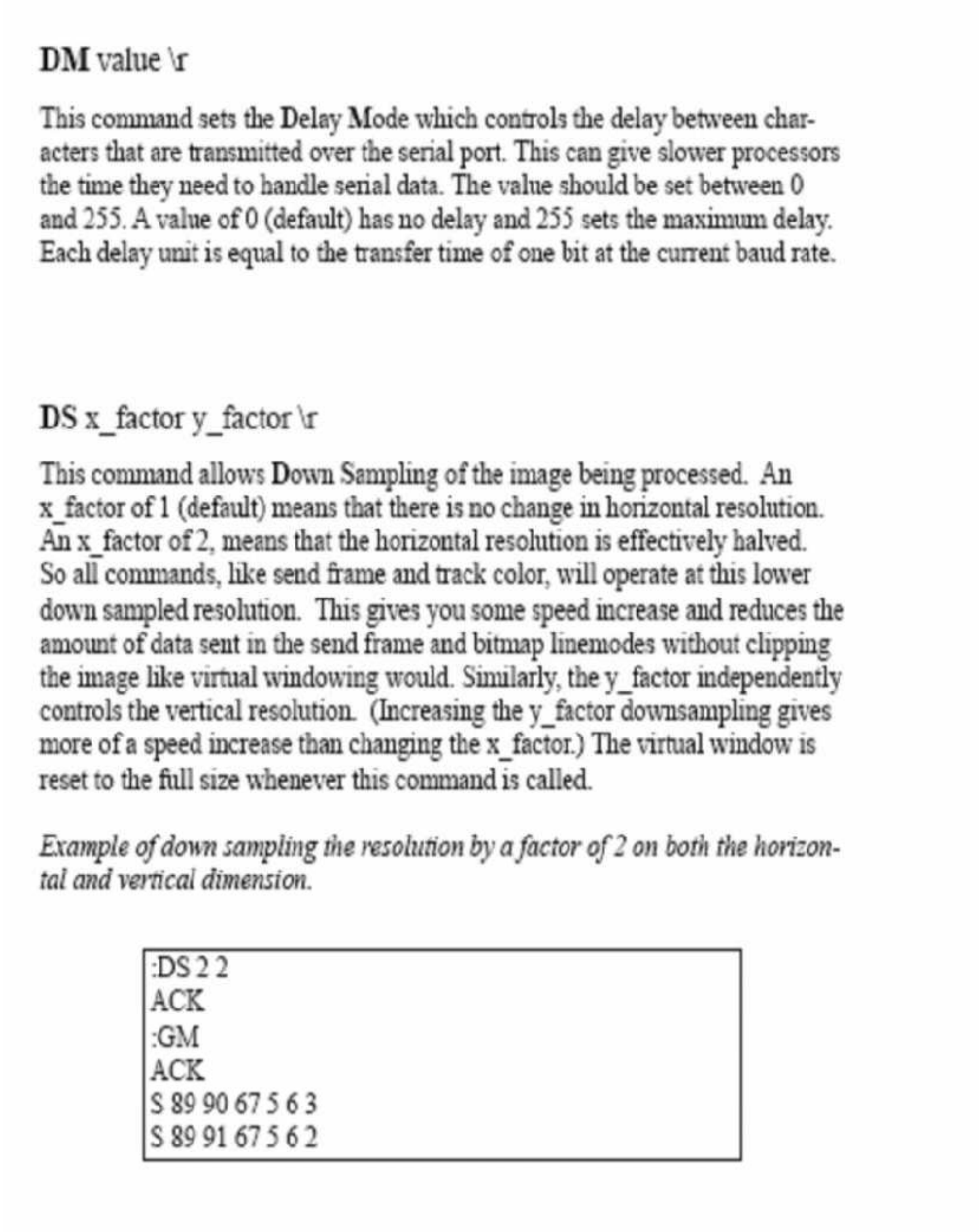
This command toggles the **C**amera module's **P**ower. A value of 0, puts the camera module into a power down mode. A value of 1 turns the camera back on while maintaining the current camera register values. This should be used in situations where battery life needs to be extended, while the camera is not actively processing image data. Images in the frame buffer may become corrupt when the camera is powered down.

CT boolean \r

This command toggles the **C**amera **T**ype while the camera is in slave mode. Since the CMUcam2 can not determine the type of the camera without communicating with the module, it is not possible for it to auto-detect the camera type in slave mode. A value of 0, sets the CMUcam2 into ov6620 mode. A value of 1 sets it into ov7620 mode. The default slave mode startup value assumes the ov6620.

DC value \r

This command sets the **C**hannel that is used for frame **D**ifferencing commands. A value of 0, sets the frame differencing commands LF and FD to use the red (Cr) channel. A value of 1 (default) sets them to use the green (Y) channel, and 2 sets them to use the blue (Cb) channel.



DM value \r

This command sets the **D**elay **M**ode which controls the delay between characters that are transmitted over the serial port. This can give slower processors the time they need to handle serial data. The value should be set between 0 and 255. A value of 0 (default) has no delay and 255 sets the maximum delay. Each delay unit is equal to the transfer time of one bit at the current baud rate.

DS x_factor y_factor \r

This command allows **D**own Sampling of the image being processed. An x_factor of 1 (default) means that there is no change in horizontal resolution. An x_factor of 2, means that the horizontal resolution is effectively halved. So all commands, like send frame and track color, will operate at this lower down sampled resolution. This gives you some speed increase and reduces the amount of data sent in the send frame and bitmap linemodes without clipping the image like virtual windowing would. Similarly, the y_factor independently controls the vertical resolution. (Increasing the y_factor downsampling gives more of a speed increase than changing the x_factor.) The virtual window is reset to the full size whenever this command is called.

Example of down sampling the resolution by a factor of 2 on both the horizontal and vertical dimension.

```
:DS 2 2
ACK
:GM
ACK
S 89 90 67 5 6 3
S 89 91 67 5 6 2
```

FD threshold \r

This command calls Frame **Differencing** against the last loaded frame using the **LF** command. It returns a type T packet containing the middle mass, bounding box, pixel count and confidence of any change since the previously loaded frame. It does this by calculating the average color intensity of an 8x8 grid of 64 regions on the image and comparing those plus or minus the user assigned *threshold*. So the larger the threshold, the less sensitive the camera will be towards differences in the image. Usually values between 5 and 20 yield good results. (In high resolution mode a 16x16 grid is used with 256 regions.)

FS boolean \r

This command sets the Frame Streaming mode of the camera. A value of 1, enables frame streaming, while a 0 (default) disables it. When frame streaming is active, a send frame command will continuously send frames back to back out the serial connection.

GB \r

This command Gets a Button press if one has been detected. This command returns either a 1 or a 0. If a 1 is returned, this means that the button was pressed sometime since the last call to Get Button. If a 0 is returned, then no button press was detected.

GH <channel> \r

This command Gets a Histogram of the *channel* specified by the user. The histogram contains 28 bins each holding the number of pixels that occurred within that bin's range of color values. So bin 0 on channel 0 would contain the number of red pixels that were between 16 and 23 in value. If no arguments are given, get histogram uses the last channel passed to get histogram. If get histogram is first called with no arguments, the green channel is used. The value returned in each bin is the number of pixels in that bin divided by the total number of pixels times 256 and capped at 255.

GI \r

This command Gets the auxiliary I/O Input values. When get inputs is called, a byte is returned containing the values of the auxiliary IO pins. This can be used to read digital inputs connected to the auxiliary I/O port. The aux I/O pins are internally lightly pulled high. See page 22 for pin numbering. Note that the pins are pulled up internally by the processor.

Example of how to read the auxiliary I/O pins. (in this case, pins 0 and 1 are high, while pins 2 and 3 are low).

```
:GI
3
ACK
:
```

GM \r

This command will Get the Mean color value in the current image. If, optionally, a subregion of the image is selected via virtual windowing, this function will only operate on the selected region. The mean values will be between 16 and 240 due to the limits of each color channel on the CMOS camera. It will also return a measure of the average absolute deviation of color found in that region. The mean together with the deviation can be a useful tool for automated tracking or detecting change in a scene. In YCrCb mode RGB maps to CrYCb.

This command returns a Type S data packet that by default has the following parameters:

S Rmean Gmean Bmean Rdeviation Gdeviation Bdeviation\r

Example of how to grab the mean color of the entire window:

```
:SW 1 1 40 143
ACK
:GM
ACK
S 89 90 67 5 6 3
S 89 91 67 5 6 2
```

GS servo \r

This command will Get the last position that was sent to the Servos.

Example of how to use get servo:

```
:GS 1
ACK
128
:
```

GT \r

This command Gets the current Track color values. This is a useful way to see what color values track window is using.

This example shows how to get the current tracking values:

```
:TW
ACK
T 12 34 ....
:GT
ACK
200 16 16 240 20 20
:
```

GV \r

This command Gets the current Version of the firmware and camera module version from the camera. It returns an ACK followed by the firmware version string. c6 means that it detects an OV6620, while c7 means that it detected an OV7620.

Example of how to ask for the firmware version and camera type:

```
:GV
ACK
CMUcam2 v1.00 c6
```

GW \r

This command Gets the current virtual Windowing values. This command allows you to confirm your current window configuration. It returns the x1, y1, x2 and y2 values that bound the current window.

HC #_of_bins scale \r

This command lets you Configure the Histogram settings. The first parameter takes one of three possible values. A value of 0 (default) will cause GH to output 28 bins. A value of 1 will generate 14 bins and a value of 2 will generate 7 bins. The scale parameter (default 0) allows you to better examine bins with smaller counts. Bin values are scaled by 2^{scale} where scale is the second parameter of the command.

#_of_bins

Input	Bins
0	28
1	14
2	7

HD boolean \r

This command enables or disables HiRes frame Differencing. A value of 0 (default) disables the high resolution frame differencing mode, while a value of 1 enables it. When enabled, frame differencing will operate at 16x16 instead of 8x8. The captured image is still stored internally at 8x8. The extra resolution is achieved by doing 4 smaller comparisons against each internally stored pixel. This will only yield good results when the background image is relatively smooth, or has a uniform color.

HR state \r

This sets the camera into HiRes mode. This is only available using the OV6620 camera module. A *state* value of 0 (default) gives you the standard 88x143, while 1 gives you 176x287. HiRes mode truncates the image to 176x255 for tracking so that the value does not overflow 8 bits.

HT boolean \r

This command enables or disables **H**istogram **T**racking. When histogram tracking is enabled, only values that are within the color tracking bounds will be displayed in the histograms. This allows you to select exact color ranges giving you more detail, and ignoring any other background influences. A value of 0 (default) will disable histogram tracking, while a value of 1 will enable it. Note that the tracking noise filter applies just like it does with the TC and TW commands.

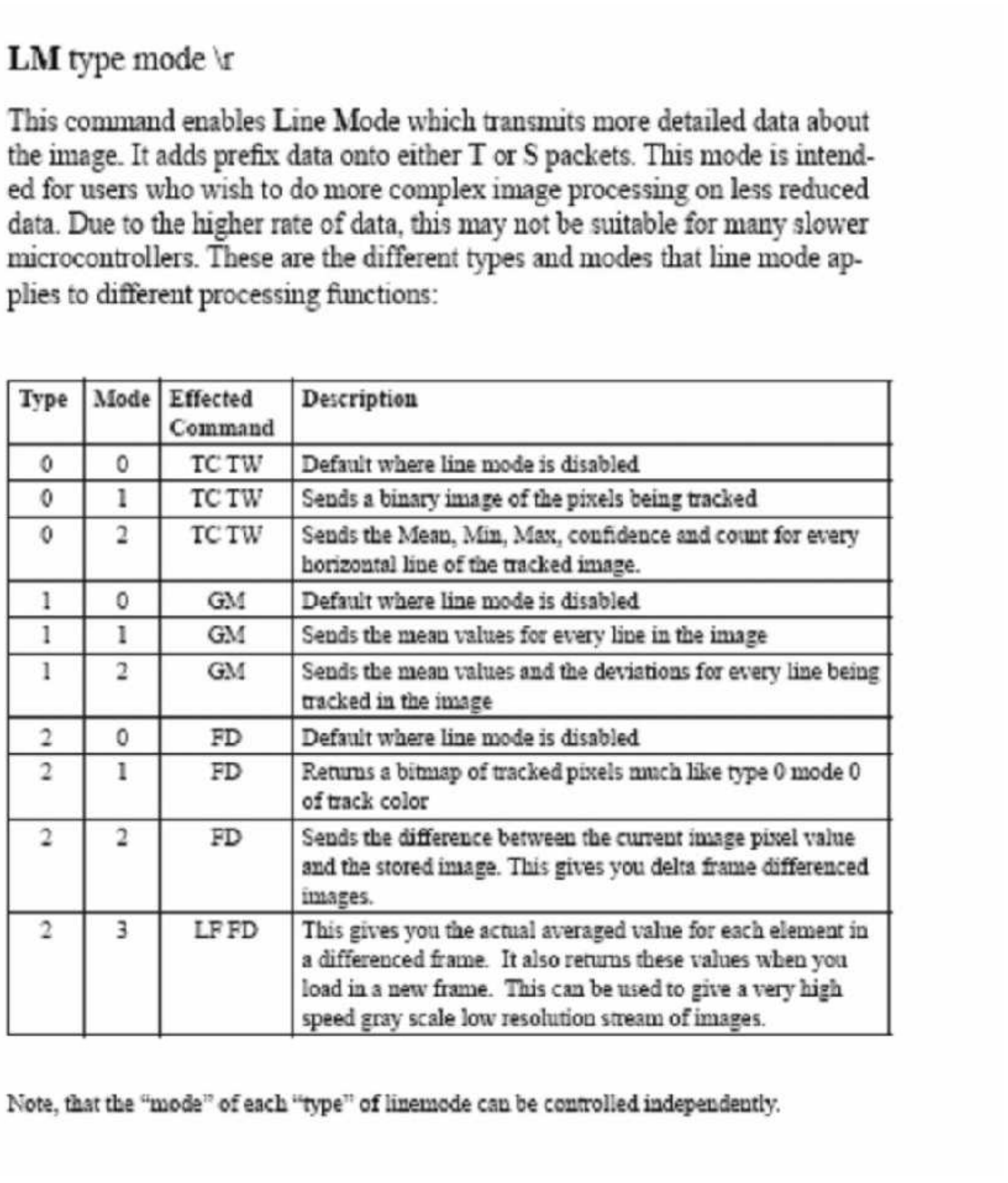
L0 boolean \r

L1 boolean \r

These commands enable and disable the two tracking **L**EDs. A value of 0 will turn the LED off, while a value of 1 will turn it on. A value of 2 (default) will leave the LED in automatic mode. In this mode, LED 1 turns on when the camera confidently detects an object while tracking and provides feedback during a send frame. In automatic mode, LED 0 does nothing, so it can be manually set.

LF \r

This command **L**oads a new **F**rame into the processor's memory to be differenced from. This does not have anything to do with the camera's frame buffer. It simply loads a baseline image for motion differencing and motion tracking.



LM type mode \r

This command enables **Line Mode** which transmits more detailed data about the image. It adds prefix data onto either T or S packets. This mode is intended for users who wish to do more complex image processing on less reduced data. Due to the higher rate of data, this may not be suitable for many slower microcontrollers. These are the different types and modes that line mode applies to different processing functions:

Type	**Mode**	**Effected Command**	**Description**
0	0	TC TW	Default where line mode is disabled
0	1	TC TW	Sends a binary image of the pixels being tracked
0	2	TC TW	Sends the Mean, Min, Max, confidence and count for every horizontal line of the tracked image.
1	0	GM	Default where line mode is disabled
1	1	GM	Sends the mean values for every line in the image
1	2	GM	Sends the mean values and the deviations for every line being tracked in the image
2	0	FD	Default where line mode is disabled
2	1	FD	Returns a bitmap of tracked pixels much like type 0 mode 0 of track color
2	2	FD	Sends the difference between the current image pixel value and the stored image. This gives you delta frame differenced images.
2	3	LF FD	This gives you the actual averaged value for each element in a differenced frame. It also returns these values when you load in a new frame. This can be used to give a very high speed gray scale low resolution stream of images.

Note, that the "mode" of each "type" of linemode can be controlled independently.

Data Packet Description

When raw mode is disabled all output data packets are in ASCII viewable format except for the F frame and prefix packets.

ACK

This is the standard acknowledge string that indicates that the command was received and fits a known format.

NCK

This is the failure string that is sent when an error occurred. The only time this should be sent when an error has not occurred is during binary data packets.

Type **F** data packet format:

```
1 Xsize Ysize 2 r g b r g b ... r g b r g b 2 r g b r g b ... r g b r g b 3
```

1 - new frame 2 - new row 3 - end of frame
RGB (CrYCb) ranges from 16 - 240
RGB (CrYCb) represents two pixels color values. Each pixel shares the red and blue.
176 cols of R G B (Cr Y Cb) packets (forms 352 pixels)
144 rows
To display the correct aspect ratio, double each column so that your final image is 352x144

Type **H** packet:

H bin0 bin1 bin2 bin3 ... bin26 bin27 \r

This is the return packet from calling get histogram (**GH**). Each bin is an 8 bit value that represents the number of pixels that fell within a set range of values on a user selected channel of the image.

Bin0 – number of pixels between 16 and 23
Bin1 – number of pixels between 24 and 31
.
.
.
Bin27 – number of pixels between 232 and 240

Type **T** packet:

```
T mx my x1 y1 x2 y2 pixels confidence\r
```

This is the return packet from a color tracking or frame differencing command.

mx - The middle of mass x value
my - The middle of mass y value
x1 - The left most corner's x value
y1 - The left most corner's y value
x2 - The right most corner's x value
y2 -The right most corner's y value
pixels –Number of Pixels in the tracked region, scaled and capped at 255: (pixels+4)/8
confidence -The (# of pixels / area)*256 of the bounded rectangle and capped at 255

Type S data packet format:

```
S Rmean Gmean Bmean Rdeviation Gdeviation Bdeviation \r
```

This is a statistic packet that gives information about the camera's view

Rmean - the mean Red or Cr (approximates r-g) value in the current window
Gmean - the mean Green or Y (approximates intensity) value found in the current window
Bmean - the mean Blue or Cb (approximates b-g) found in the current window
R*deviation* - the *deviation of red or Cr found in the current window
G*deviation*- the *deviation of green or Y found in the current window
B*deviation*- the *deviation of blue or Cb found in the current window

*deviation: The mean of the absolute difference between the pixels and the region mean.