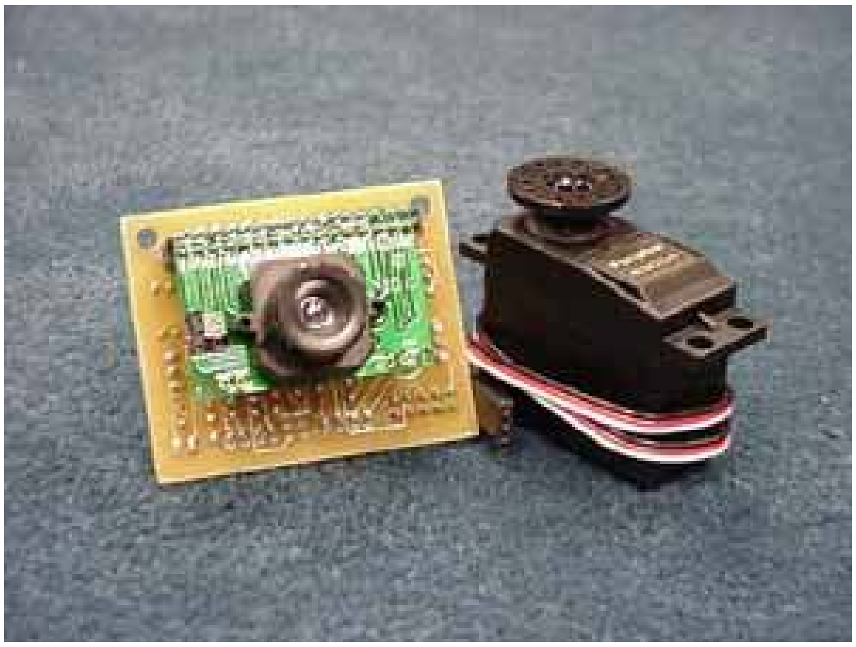
CMUcam SDK Documentation

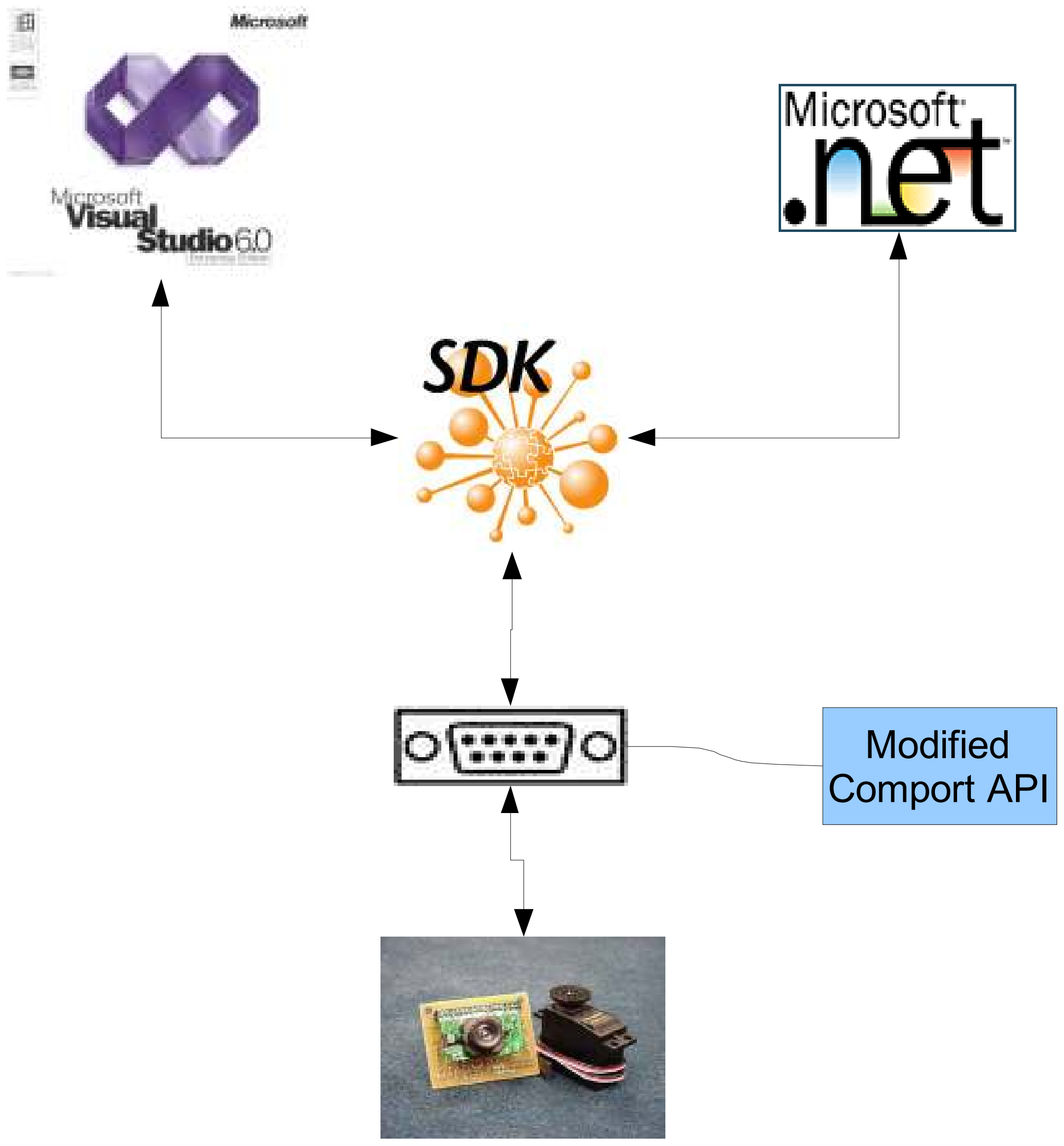


Developed By: Michael Konicki

Index

Introduction

The CMU.DLL is designed to serve as a SDK for various languages. It provides all the comport and camera functions nessecary to make any program quickly and easily using the CMUcam. It works in a variety of languages across the MS Visual Studio platform (code examples come in VB and C++). Sample code (C++ and VB) as well as full documentation on all of the functiosn within the CMU.DLL are available in this document.



NOTE: This dll can not be loaded onto the 4W4T robot directly and will require a computer to communicate between the two.

ComPort Functions

The Comport functions were initially developed by Karl Riehl in 2005. They have been slightly modified to allow for ease of use for the users. The Comport handle is now a global variable, which means it no longer has to be passed around with every function call. The comport initialization routines have been reconfigured to automatically setup to match the CMUCam's default connection settings. Several other minor changes were introduced to make things work smoothly with the CMUCam

While these functions are left public for debugging or advanced users they should not be used unless you know what you are doing. (**NOTE:** This means that the Comport functions are no longer generic and if you wish to use comport functions you should revert to Karl's original API).

CMU_SetComSettings(int BaudRate, int ByteSize, DWORD Parity, DWORD StopBits)

This function sets the Comport settings so that it is properly configured for the CMUcam to prevent the user from having to do this manually.

To set the settings manually the CMUCam uses:

BaudRate: 115200
ByteSize: 8
Parity: None
StopBits: 1
FlowControl: None

Usage:

```
//This will set the Comport to the default CMUCam settings
      if(!CMU_SetComSettings(115200, 8, NOPARITY, ONESTOPBIT))
     cout << "Error Occurred" << endl;
```

CMU_SetTimeouts(DWORD ReadInterval, int ReadMultiplier, int ReadConstant, int WriteMultiplier, int WriteConstant)

This function sets the timeouts for:

ReadInterval → Maximum time allowed to elapse between the arrival of two bytes on the communications line, in milliseconds.

ReadMultiplier → Multiplier used to calculate the total time-out period for read operations, in milliseconds.

ReadConstant → Constant used to calculate the total time-out period for read operations, in milliseconds.

WriteMultiplier → Multiplier used to calculate the total time-out period for write operations, in milliseconds.

WriteConstant → Constant used to calculate the total time-out period for write operations, in milliseconds.

Usage:

```
      //This sets the timeouts to their default values
      if(!CMU_SetTimeouts( MAXWORD, 0 ,0 ,0 ,0 ))
           cout << "Error Occurred" << endl;
```

CMU_ClearOutputBuffer()

The function clears the output buffer.

Usage:

```
//Clears the output buffer
     If(!CMU_ClearOutputBuffer())
          cout << "Error Occurred" << endl;
```

CMU_ClearInputBuffer()

The function clears the input buffer.

Usage:

```
     //Clears the input buffer
     If(!CMU_ClearInputBuffer())
          cout << "Error Occurred" << endl;
```

CMU_ClearBuffers()

The function clears the output and input buffers.

Usage:

```
     //Clears the output and input buffers
     If(!CMU_ ClearBuffers ())
          cout << "Error Occurred" << endl;
```

CMU_SetBuffer(int Incoming, int Outgoing)

The function sets the read and write buffer sizes to *Incoming & Outgoing*.

Usage:

```
     //Sets the buffer sizes to 1028 bytes each
     If(!CMU_ SetBuffer(1028,1028))
          cout << "Error Occurred" << endl;
```

CMU_ClearWrite()
The function clears any waiting writes from the buffer.

Usage:
```
     //Clears any data in the write buffer
     If(!CMU_ ClearWrite())
          cout << "Error Occurred" << endl;
```

CMU_ClearRead()
The function clears any waiting reads from the buffer.

Usage:
```
//Clears any data in the read buffer
     If(!CMU_ ClearRead())
          cout << "Error Occurred" << endl;
```

CMU_ClearReadWrite()
The function clears any waiting read or writes from the buffer.

Usage:
```
     //Clears any data in the write and read buffers
     If(!CMU_ ClearReadWrite())
          cout << "Error Occurred" << endl;
```

CMU_ClearSignal()
The function lets the port know that is not available to send and receive data.

Usage:
```
     //Clears the signals which sets the port to unavailable
     If(!CMU_ ClearSignal())
          cout << "Error Occurred" << endl;
```

CMU_SetSignal()
The function lets the port know that it is available to send and receive data.

Usage:
```
     //Sets the signals which sets the port to available
     If(!CMU_ SetSignal())
          cout << "Error Occurred" << endl;
```

CMU_PortOpen(char *Port)
The function opens the comm port and sets the port to default settings.

Usage:

```
      //Opens the port pointed to by char* Port
      If(!CMU_ PortOpen("COM1"))
            cout << "Error Occurred" << endl;
```

CMU_ClosePort()
This function closes the comm port.

Usage:

```
      //Closes the port
      CMU_ ClosePort()
```

CMU_Read(unsigned char* ch, int NumBytesToRead)
The function reads in from the input buffer and places the first byte at the address that is pointed to by the unsigned char pointer `ch`.

Usage:

```
      //Reads into buff[] buffer bytes
      //int this example it reads 30 bytes off of the buffer
      //and places them in buff[]
      Char buff[30];
      Int buffer 30;

      if(!CMU_Read(&buff[0],buffer)) ())
            cout << "Error Occurred" << endl;
```

CMU_Write(unsigned char* ch, int NumBytesToWrite)
The function writes the number of bytes starting at the memory location specified by the unsigned char pointer to the port `ch`.

Usage:

```
//Writes into the write buffer, buff[] of size bytes
      //int this example it writes 30 bytes into the Comport's
      //write buffer
      Char buff[30];
      Int size 30;

      if(!CMU_Read(&buff[0],size)) ())
            cout << "Error Occurred" << endl;
```

CMU_InputBufferSize()
The function returns the number of bytes that are currently in the input buffer.

Usage:
```
    //Returns the number of bytes in the Comport's input buffer
    !if(!CMU_InputBufferSize())
         cout << "Error Occurred" << endl;
```

CMU_OutputBufferSize()
The function returns the number of bytes that are currently in the output buffer.

Usage:
```
    //Returns the number of bytes in the Comport's output
    //buffer
    !if(!CMU_OnputBufferSize())
         cout << "Error Occurred" << endl;
```

INTERNAL Functions

The INERNAL functions handle all of the reading and processing of data from the CMUCam. They should never be called in a program as they are called by the functions already. They do return the full message from the CMUCam for advanced debugging purposes.

int _stdcall INTERNAL_ACKNCK(unsigned char * str, int buffer)
This function is called after a function does a `CMU_Write`. It waits for a ComEvent, which is fired when something enters the input buffer. It then reads buffer number of bytes (as the length of the return message can vary among the different commands) and stores it in str (which is for debuging purposes, no strings are returned to the users program, it is simply for dll development and testing). After the message is received it checks to see whether the message contains and ACK or NCK from the CMUCam. If it is and ACK it returns a 1, for a NCK it returns a 0.

Usage:
```
     Char ret[30]

     //Write to CMUcam here

     //Get the reply from the CMUCam and ensure it was an ACK
     //else return an error
     if(!INTERNAL_ACKNCK(ret, 5))
          cout << "Error Occurred" << endl;
```

int _stdcall INTERNAL_PACKET_PROCESS(int* slot1, int* slot2, int* slot3, int* slot4, int* slot5, int* slot6, int* slot7, int* slot8, int* slot9, int buffer)

This function is called in place of `INTERNAL_ACKNCK()` when there is going to be a stream of packets coming in from the CMUCam. It will read buffer bytes from the stream and then process it for a packet. Based on the way the CMUCam sends data this function splits the stream on a `Chr(32)` or space and will keep splitting until it hits a `Chr(13)` or carriage return since this character causes strtok to fail. The function then stores the numbers it got from the stream into the `int* slot(1-9)`. Depending on the packet type some of these values are not used but the function that called this one knows which values are to be used, and which are to be ignored. Those functions also know what the numbers mean. By leaving the function generic it works for all (M, N, C, S) packets.

Usage:

```
	//Temp variables that get ignored
	int temp1,temp2,temp3;
	//Variables that contain the good data from the stream
	int X1, Y1, X2, Y2, pixels, confidence;

	//Get the data from the stream, have to send all 9
	//ints but we are only using 6 in this example because
	//we are expecting a C-Packet
	if(!INTERNAL_PACKET_PROCESS(&X1,&Y1,&X2,&Y2,&pixels,
		&confidence,&temp1,&temp2,&temp3, 50))
		cout << "Error Occurred" << endl;

	//At this point the data would be filled into a struct or
	//if using the function directly it could be stored or
	//directly referenced from the ints
```

CMUCam Functions

These functions provide access to all of the CMUCams different abilities. They are designed to provide ease of use across the MS Studio platform. They are also designed to be open for the user, meaning that they are not to restrictive and allow the user to accomplish the solutions in the manner he/she would normally do.

CAM_Ping ()

This function sends a `'\r'` command to the CMUCam which returns an ACK if the command is understood.

Usage:

```
     //See if the CMUCam is working
     if(!CAM_Ping())
          cout << "Error Occurred" << endl;
```

CAM_Init(char * port)

This function initializes the comport, `char* port`, with the default CMUCam settings.

Usage:

```
     //Init CMUCam on COM1
     if(!CAM_Init("COM1"))
          cout << "Error Occurred" << endl;
```

CAM_TrackLight(unsigned char mode)

This function sets the green LED on the CMUCam to off,on,auto. Auto mode is when the light turns on when the object the camera is tracking is found within the CMUCams view. This function is mainly used as a debugging utility since it is easy to see if the camera is working with the green light going.

Usage:

```
     //Turn the TrackLight on (off = MODE_OFF, auto = MODE_AUTO)
     if(!CAM_TrackLight(MODE_ON))
          cout << "Error Occurred" << endl;
```

CAM_Brightness(int bri)

This function sets the brightness level of the CMUCam. The values can range from 0 -255.

Usage:

```
     //Set the brightness to 15
     if(!CAM_Brightness(15))
          cout << "Error Occurred" << endl;
```

CAM_ClockSpeed(unsigned char speed)
This function sets the clock speed on the CMUCam to either 17, 13, 11, 9, 8, 7, or 6 frames per second. This is not the streaming FPS rating, but how many frames the CMUCam internally processes.

Usage:

```
     //Turn the clockspeed to 17
     if(!CAM_ClockSpeed(FPS_17))
          cout << "Error Occurred" << endl;
```

CAM_Contrast(int con)
This function sets the contrast level of the CMUCam. The values can range from 0 -255.

Usage:

```
     //Set the contrast to 150
     if(!CAM_Contrast(150))
          cout << "Error Occurred" << endl;
```

CAM_GetVersion(unsigned char * version)
This function returns the version of the firmware loaded on the CMUCam into the string pointed to by `version`.

Usage:

```
     Char ver[20];

     //Get and display the version of the firmware
     if(!CAM_GetVersion(ver))
          cout << "Error Occurred" << endl;
     else
          cout << "Current Version: " << ver << endl;
```

CAM_HalfHorizontalMode(unsigned char mode)
This function sets half-horizontal mode on and off. On means that every other column is skipped in processing which makes the camera run much faster. Off means the CMUCam processes every column in an image.

Usage:

```
     //Set HH-Mode to off (MODE_ON = on)
     if(!CAM_HalfHorizontalMode(MODE_OFF))
          cout << "Error Occurred" << endl;
```

CAM_NoiseFilter(unsigned char filter)
This function sets the noise filter on and off.

Usage:
```
	//Set NoiseFilter to off (MODE_ON = on)
	if(!CAM_NoiseFilter(MODE_OFF))
		cout << "Error Occurred" << endl;
```

CAM_MiddleMassMode(int mode)
This function sets the middlemass mode on, off, and on with servo support. When the MM-Mode is set to on the packets returning from streams will contain two extra values that give the centroid data. This function is automatically called when initializing the streams so users should never need to use it.

Usage:
```
	//Set middlemass to off (MODE_ON = on)
	if(!CAM_MiddleMassMode(MODE_OFF))
		cout << "Error Occurred" << endl;
```

CAM_Reset()
This function resets all the registers of the CMUCam, restores it to default settings.

Usage:
```
	//reset the camera
	if(!CAM_Reset())
		cout << "Error Occurred" << endl;
```

CAM_SwitchMode(unsigned char mode)
This function sets the switch mode on and off. When switch mode is on every other packet is sent as an S-Packet, which is a data packet. This means the packets come in like M/C, S, M/C, S, M/C, S,..... This function should never be called except for advanced users, when streams are needed with alternating packets there are functions set to do this automatically.

Usage:
```
	//Set swithmode to off (MODE_ON = on)
	if(!CAM_SwitchMod(MODE_OFF))
		cout << "Error Occurred" << endl;
```

CAM_EndPacket()

This function stops a packet stream and sets the CMUCam back into a state were it can receive new instructions. This function also clears out the buffers automatically and it also checks to make sure the packet stream has died.

Usage:

```
      //End a packet stream
      if(!CAM_EndPacket())
            cout << "Error Occurred" << endl;
```

CAM_STR_MC()

This function initializes the mean color packet stream. This stream returns the mean and deviation values of R, G, and B colors. To read the packets use the `CAM_Get_MC()` function

Usage:

```
      //Initialize a mean color stream
      if(!CAM_STR_MC())
            cout << "Error Occurred" << endl;
```

CAM_Get_MC(S_PACKET* spack)

This function is called after a mean color packet stream has been started. It will rip a S-Packet out of the stream and return it to the `spack` pointer.

Usage:

```
      S_PACKET spack;

      ..start stream here..

      //Get the packet data from the stream
      if(!CAM_Get_MC(spack))
            cout << "Error Occurred" << endl;
      else
            cout << <spack data members> << endl;
```

CAM_STR_TCO_Centroid()

This function initializes the track center object stream. This stream will turn on centroid data before starting.

Usage:

```
      //Initialize a TCO stream with centroid data
      if(!CAM_STR_TCO_Centroid())
            cout << "Error Occurred" << endl;
```

CAM_Get_TCO_Centroid(M_PACKET* mpack)
This function is called after a track center object packet stream has been started. It will rip a M-Packet out of the stream and return it to the `mpack` pointer.

Usage:
```
     M_PACKET mpack;

     ..start stream here..

     //Get the TCO packet w/ centroid data from the stream
     if(!CAM_Get_TCO_Centroid(mpack))
          cout << "Error Occurred" << endl;
     else
          cout << <mpack data members> << endl;
```

CAM_STR_TCO()
This function initializes the track center object stream. This stream will turn off centroid data before starting.

Usage:
```
     //Initialize a TCO stream without centroid data
     if(!CAM_STR_TCO())
          cout << "Error Occurred" << endl;
```

CAM_Get_TCO(C_PACKET* cpack)
This function is called after a track center object packet stream has been started. It will rip a C-Packet out of the stream and return it to the `cpack` pointer.

Usage:
```
     C_PACKET cpack;

     ..start stream here..

     //Get the TCO packet w/o centroid data from the stream
     if(!CAM_Get_TCO(cpack))
          cout << "Error Occurred" << endl;
     else
          cout << <cpack data members> << endl;
```

CAM_STR_TCO_Servo()
This function initializes the track center object stream. This stream will turn on centroid data and servo support before starting.

Usage:
```
     //Initialize a TCO stream with centroid and servo data
     if(!CAM_STR_TCO_Servo())
          cout << "Error Occurred" << endl;
```

CAM_Get_TCO_Servo(N_PACKET* npack)
This function is called after a track center object packet stream has been started. It will rip a C-Packet out of the stream and return it to the `npack` pointer.

Usage:
```
     N_PACKET npack;

     ..start stream here..

     //Get the TCO packet w centroid and servo data
     if(!CAM_Get_TCO(npack))
          cout << "Error Occurred" << endl;
     else
          cout << <npack data members> << endl;
```

CAM_ColMode_RGB_ON()
This function sets the CMUCam to RGB mode with auto white balancing on.

Usage:
```
     //set CMUCam to RGB with white balance on
     if(!CAM_ColMode_RGB_ON())
          cout << "Error Occurred" << endl;
```

CAM_ColMode_RGB_OFF()
This function sets the CMUCam to RGB mode with auto white balancing off.

Usage:
```
     //set CMUCam to RGB with white balance off
     if(!CAM_ColMode_RGB_OFF())
          cout << "Error Occurred" << endl;
```

CAM_ColMode_YCrCb_ON()
This function sets the CMUCam to YCrCb mode with auto white balancing on.

Usage:
```
      //set CMUCam to YCrCb with white balance on
      if(!CAM_ColMode_YCrCb_ON())
           cout << "Error Occurred" << endl;
```

CAM_ColMode_YCrCb_OFF()
This function sets the CMUCam to YCrCb mode with auto white balancing off.

Usage:
```
      //set CMUCam to YCrCb with white balance off
      if(!CAM_ColMode_YCrCb_OFF())
           cout << "Error Occurred" << endl;
```

CAM_AutoExposure_GainOFF()
This function sets the CMUCam's auto exposure/auto gain to off

Usage:
```
      //set CMUCam's auto exposure off
      if(!CAM_AutoExposure_GainOFF())
           cout << "Error Occurred" << endl;
```

CAM_AutoExposure_GainON()
This function sets the CMUCam's auto exposure/auto gain to on

Usage:
```
      //set CMUCam's auto exposure on
      if(!CAM_AutoExposure_GainON())
           cout << "Error Occurred" << endl;
```

CAM_STR_TCC_Centroid(COL_PACKET* cpack)
This function initializes the track custom color stream. It is feed its custom colors by having the user fill out a COL_PACKET and send it into the function. This stream has centroid data turned on.

Usage:

```
     //Initialize a TCC stream using the users colors and
     //centroid data
     COL_PACKET = cp;
     cp.Rmax = 100;
     cp.Rmin = 0;
     cp.Bmax = 255;
     cp.Bmin = 0;
     cp.Gmax = 0;
     cp.Gmin = 0;

     if(!CAM_STR_TCC_Centroid(cp))
          cout << "Error Occurred" << endl;
```

CAM_Get_TCC_Centroid(M_PACKET* mpack)
This function is called after a track custom color packet stream has been started. It will rip a M-Packet out of the stream and return it to the `mpack` pointer.

Usage:

```

     M_PACKET mpack;

     ..start stream here..

     //Get the TCO packet w centroid and servo data
     if(!CAM_Get_TCC_Centroid(mpack))
          cout << "Error Occurred" << endl;
     else
          cout << <mpack data members> << endl;
```

CAM_STR_TCC(COL_PACKET* cpack)
This function initializes the track custom color stream. It is feed its custom colors by having the user fill out a COL_PACKET and send it into the function.

Usage:
```
      //Initialize a TCC stream using the users colors
      COL_PACKET = cp;
      cp.Rmax = 100;
      cp.Rmin = 0;
      cp.Bmax = 255;
      cp.Bmin = 0;
      cp.Gmax = 0;
      cp.Gmin = 0;

      if(!CAM_STR_TCC(cp))
           cout << "Error Occurred" << endl;
```

CAM_Get_TCC(C_PACKET* mpack)
This function is called after a track custom color packet stream has been started. It will rip a C-Packet out of the stream and return it to the `cpack` pointer.

Usage:
```
      C_PACKET cpack;

      ..start stream here..

      //Get the TCO packet w centroid and servo data
      if(!CAM_Get_TCC(cpack))
           cout << "Error Occurred" << endl;
      else
           cout << <cpack data members> << endl;
```

CAM_DumpFrame(unsigned char* FRAME)
This function will dump a frame from the camera to the string pointed to by FRAME. This process, depending on frame resolution, will take several seconds. To process this data, which is an F-Packet, it is important to note that:

1 – New Frame
2 – New Col
3 – End of Frame
34407 – Default Frame Size (Max Resolution)
RGB (YcrCb) ranges from 16 – 255

Example:

12rgbrgb...rgb2rgbrgb....rgb2rgbrgb.....rgbrgb3

Usage:

```
    char FRAME[FRAME_SIZE];

    //Dump a frame
    if(!CAM_DumpFrame(FRAME))
         cout << "Error Occurred" << endl;

    ...process frame here... (see example programs)
```

CAM_ClosePort()
This function closes the comm port by calling CMU_ClosePort.

Usage:

```
    //Closes the port
    CAM_ ClosePort()
```

CAM_SetWindowSize(int x, int y, int x2, int y2)
This function allows the user to adjust the resolution size of the CMUCam. The default settings are 1, 1, 80, 143. x values ranger from 1 – 80, y values range from 1 – 143.

Usage:

```
    //adjust the CMUCam's resolution
    if(!CAM_SetWindowSize(1,1,80,143)
```

Using CMU.DLL in C++

Start a new project or open a file you wish to use. It is recommended, although not nessecary that you use the header file (CMU.H) since all the functions and typedefs are declared for you. If you have it now is the time to add it to your project. If you don't add the function prototyopes from the dll you plan on using.

example: `extern "C" int _stdcall CAM_ClosePort()`

After adding all the functions you wish to use you should now be able to compile the project when you use those functions. Make sure that the dll is in the project folder or else you will get LNK2001 errors.

It is also recommended that you include the CMU.lib file to your project. This will make debugging much easier as you are allowed to step through the code and see what is actually going on.

Using CMU.DLL in VB

Open a new project and add a new module to your project. Now add to the module all the functions you wish to use from the dll, examples:

```
'Close port (normal example)
Public Declare Function CAM_ClosePort Lib "CMU.dll" () As Long

'Initialize port and CMUCam (string example)
Public Declare Function CAM_Init Lib "CMU.dll" (ByVal port As
String) As Long

'DumpFrame (unsigned char* example)
Public Declare Function CAM_DumpFrame Lib "CMU.dll" (ByVal frame
As Long) As Long
```

Now you can call the functions in your VB forms like nomral functions. Again make sure you have the dll in a place that VB can find it or you will get missing file errors.

To ease debugging you can set VB to debugging mode. Project Properties → Debugging tab. This will allow you to step through some of the dll code which makes it easier to find errors

Example Programs

Green → Comments
Red → DLL related (function calls, declerations)
Blue → Important concepts

LightShow → C++

Explanation

The purpose of this program is to show basic functions in the dll and how to connect to the CMUCam using the dll functions. All this program does is flicker the tracking light on the camera on and off 25 times and then exits. The program is fully documented to learn from.

Image(s)

None

Source Code

```
//Includes
#include <windows.h>
#include <iostream.h>

//Functions in the CMU.DLL
extern "C" int _stdcall CAM_Init(char *port);
extern "C" void _stdcall CAM_ClosePort();
extern "C" int _stdcall CAM_TrackLight(unsigned char mode);
extern "C" int _stdcall CAM_Ping();

void main()
{
      int i;      //Used as a loop counter

      //Initialize the comport and setup CMUCam
      CAM_Init("COM1");

      //Do a ping test to see if the CMUCam is hooked and
      //respoding, exit the program if aint working
      if(!CAM_Ping())
      {
            cout << "Ping Error" << endl;
            Sleep(500);
            return;
      }

      for(i=0;i<25;i++)
      {
            //Turn on the light
            if(!CAM_TrackLight('1'))
                  cout << "ERROR" << endl;
            else
                  cout << "Light On" << endl;
            //Delay so the user can see the light flicker
            Sleep(300);

            //Turn off the light
            if(!CAM_TrackLight('0'))
                  cout << "ERROR" << endl;
            else
                  cout << "Light Off" << endl;

            //Delay so the user can see the light flicker
            Sleep(300);

      }

      cout << "And that concludes the light show, closing port and shuting
               down" << endl;

      //Shutdown the CMUCam's comport
      CAM_ClosePort();


      Sleep(4000);
      return;
}
```

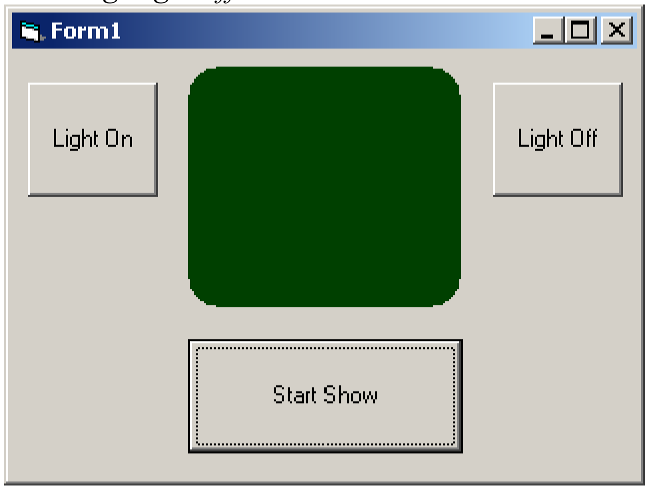
LightShow → VB

Explanation

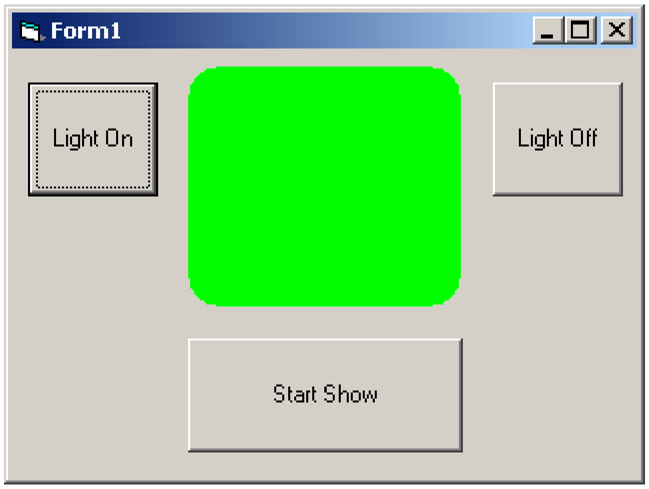
Similar to the program above but in VB 6.

Image(s)

Tracking Light off



Tracking Light On



Source Code

Module Code

```
'Close port
Public Declare Function CAM_ClosePort Lib "C:\CMU.dll" () As Long

'Initialize port and CMUCam
Public Declare Function CAM_Init Lib "C:\CMU.dll" (ByVal port As String) _
As Long

'TrackLight
Public Declare Function CAM_TrackLight Lib "C:\CMU.dll" (ByVal ByteArr As _
Byte) As Long

'Ping for testing purposes
Public Declare Function CAM_Ping Lib "C:\CMU.dll" () As Long

'Sleep function
Public Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
```

Form Code

```
Private Sub Form_Load()
    'Open the comport for business
    CAM_Init ("COM1")
End Sub

Private Sub Form_Unload(Cancel As Integer)
    'shutdown the comport
    CAM_ClosePort
End Sub

Private Sub lightclick_Click()
    Dim t As Integer
    Dim i As Integer

    'Loop through and change the color
    For i = 0 To 15
        'turn the light on and show that on the form
        CAM_TrackLight (Asc("1")) 'Call the dll function to turn the light on
        sh1.BackColor = RGB(0, 255, 0)
        Me.Refresh  'Refresh the form so the shape changes color

        'Delay a little so you can see the change
        Sleep (500)

        'Turn the light off and show it on the form
        CAM_TrackLight (Asc("0"))'Call the dll function to turn the light off
        sh1.BackColor = RGB(0, 75, 0)
        Me.Refresh     'Refresh the form so the shape changes color

        'Delay a little to see the change
        Sleep (500)

    Next i

    MsgBox "Demo Done"
End Sub

Private Sub lightoff_Click()
    'Turn the light off and show it on the form
    CAM_TrackLight (Asc("0"))
    sh1.BackColor = RGB(0, 75, 0)
    Me.Refresh
End Sub

Private Sub Lighton_Click()
    'Turns the light on and updates the form
    CAM_TrackLight (Asc("1"))
    sh1.BackColor = RGB(0, 255, 0)
    Me.Refresh
End Sub
```

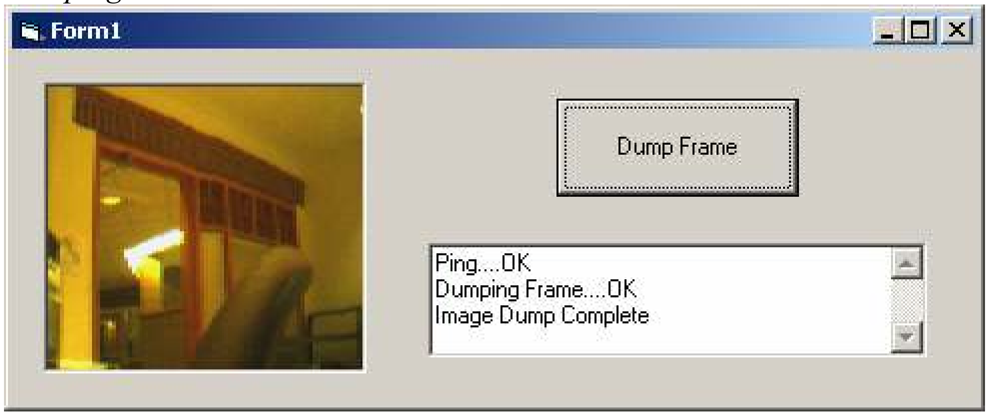
DumpFrame → VB

Explanation

This program dumps a frame from the camera by using the dll for all the function calls.

Image(s)

Dumping a Frame



```
Ping....OK
Dumping Frame....OK
Image Dump Complete
```

Source Code

Module Code

```
'Close port
Public Declare Function CAM_ClosePort Lib "C:\CMU.dll" () As Long

'Initialize port and CMUCam
Public Declare Function CAM_Init Lib "C:\CMU.dll" (ByVal port As String) _
As Long

'DumpFrame
Public Declare Function CAM_DumpFrame Lib "C:\CMU.dll" (ByVal frame As Long)_
As Long

'Ping for testing
Public Declare Function CAM_Ping Lib "C:\CMU.dll" () As Long
```

Form Code

```
Option Explicit


Private Sub DF_Click()
    Dim fr(40000) As Byte     'Byte array that holds the image
    Dim i As Integer          'catches return values from dll functions
    Dim started As Boolean    'image processing
    Dim col As Integer        'image processing
    Dim row As Integer        'image processing
    Dim x As Long             'image processing

    'Ping the camera before we try a dumpframe
    'to make sure everything is hooked up right

    i = CAM_Ping 'Ping the camera

    'Update the text box
    If i = 1 Then
        Text1.Text = "Ping....OK" & vbCrLf
        Text1.Refresh
    Else
        Text1.Text = "Ping....FAILED" & vbCrLf
        Text1.Refresh
    End If

    'Update the textbox
    Text1.Text = Text1.Text & "Dumping Frame...."
    Text1.Refresh

    'Using VB pointers get our frame
    i = CAM_DumpFrame(VarPtr(fr(0)))

    'Update the textbox
    If i = 1 Then
        Text1.Text = Text1.Text & "OK" & vbCrLf
        Text1.Refresh
    Else
        Text1.Text = Text1.Text & "FAILED" & vbCrLf
        Text1.Refresh
    End If

    col = 0 'used for image display
    row = 0 'used for image display
    started = False

    Pic.ScaleMode = vbPixels

    For x = 1 To 34407
        '1 = start of frame
        If fr(x) = 1 Then
            Pic.Cls
            started = True
        '2 = new col
        ElseIf fr(x) = 2 Then
            col = col + 2
            row = 0
        '3 = end of frame
        ElseIf fr(x) = 3 Then
           Text1.Text = Text1.Text & "Image Dump Complete"
            Exit Sub
        Else
            'Make sure we started by seeing a 1
            'before adding stuff to the picture box
            If started = True Then
                'have to double up the columns
                Pic.PSet (col, row), RGB(fr(x), fr(x + 1), fr(x + 2))
                Pic.PSet (col + 1, row), RGB(fr(x), fr(x + 1), fr(x + 2))
                row = row + 1   'go to next row
                x = x + 2       'bump x since data is stored as rgb we ripped
```

```
                                '3 off so 2 here one at the next x = 3
            End If
        End If
    Next x

End Sub

Private Sub Form_Load()
    'open and initialize the CMUCam and the Comport
    CAM_Init ("COM1")
End Sub

Private Sub Form_Unload(Cancel As Integer)
    'close out the comport
    CAM_ClosePort
End Sub
```

Usefull Concepts

This section will discuss much of the usefull concepts and development ideas associated with the CMUCam and the CMU.DLL.

CMUCam

Looping Input → When dealing with the CMUCam it is important to note that some commands, such as MC, TW, TC, actually start the CMUCam in a data dumping loop. This means that once these commands are received the CMUCam will continuosly dump packets down the comport. Any command the user sends will generally not get received as the CMUCam can only handle communications one-way on the comport. However that command that is sent during the loop will also break the loop as the CMUCam tries to grab the command, hence stopping the loop to do so. Unfortunatly that command will never read the camera. Therefore it is important to understand that if you start a loop either in the CMU.DLL or in your own programs it is important to end the loop using a '\r' or ping command. Then allow the CMUCam to sit for a moment and then send the comman you wish the CMUCam to respond to.

CMU.DLL EndPacket example is how the dll shutsdown a packet loop:

```
	if(!CAM_Ping())	//May not succeed on first try due to the
			//massive amount of incoming data
	{
		Sleep(400);	//let the comport clean itself...
		if(!CAM_Ping())	//and try again
			return 0;
		else
			return 1;
	}
	else
	{
		return 1;	//If it did succed the first time return ok
	}
```

Processing a frame dump → Another complex requirement of the CMUCam is processing a frame dump from the CMUCam. When you call the DumpFrame in the CMU.DLL or issue a "DF" directly you are going to get a massive string of data. It is important to know to process this string. The F-Packet, which is what the frame is, comes in as: 1,2,r,g,b…,r,g,b,2,r,g,b,….,r,g,b,3.

The Chr(1) signals a new frame, this is where you should start processing and displaying the following RGB values. Data before this marker can include the ACK message and sometimes some other junk data.

The Chr(2) signals a new column at this point when dumping an image you need to advance your column counter

The Chr(3) signals the end of the frame, at this point you can exit the loop you are in for the frame processing.

Example Code of Frame Processing (VB)

```
   For x = 1 To 34407    '344007 is the max size a frame can be
         '1 = start of frame
         If fr(x) = 1 Then
             Pic.Cls
             started = True
         '2 = new col
         ElseIf fr(x) = 2 Then
             col = col + 2       'since the CMUCam expects you to double
                                 'up columns bump counter by 2
             row = 0
         '3 = end of frame
         ElseIf fr(x) = 3 Then
            Text1.Text = Text1.Text & "Image Dump Complete"
             Exit Sub
         Else
             'Make sure we started by seeing a 1
             'before adding stuff to the picture box
             If started = True Then
               'have to double up the columns use (col + 1)
               Pic.PSet (col, row), RGB(fr(x), fr(x + 1), fr(x + 2))
               Pic.PSet (col + 1, row), RGB(fr(x), fr(x + 1), fr(x + 2))
               row = row + 1    'go to next row
               x = x + 2        'bump x since data is stored as rgb we
                                'ripped 3 off so 2 here one at the next x
                                '= 3
             End If
         End If
     Next x                     'advance x for the 3rd time (rgb 3 bumps)
```