GETTING STARTED, (and finished), WITH THE CMUCam2 VISION SENSOR

The following process is intended to assist you in getting the CMUCam2 from the package to an operational sensor on your Robot. It assumes you have your Robot is assembled, wired, and ready to go. If your Robot is not yet ready for use then you can perform this process with the Robot control system "on the bench".

This document is will take you through the process in the following order:

- The items required.
- Parts modification and assembly.
- Connections and Installation.
- Testing and Calibration.

In addition you will need to have, and be familiar with the following:

1. The MP.Lab CBOT compiler, (supplied in the Innovation First Control system box).
   Specifically you will need to know how to:
   - Open Projects.
   - Open files within the project.
   - Edit a file.
   - Compile a program.

2. The Innovation First IFI_Loader v1.0.10 (zip, 11-17-2004), (Available at; http://www.innovationfirst.com/).
   Specifically you will need to:
   - Connect the computer to the Robot Controller.
   - Find the .hex file.
   - Download the file to the Robot Controller.

3. The CMUcam2_fe Manual, (Supplied with this documentation).
   Specifically you will need to become familiar with the General Information and Hardware sections of the manual.

4. CMUcam2GUI_fe Folder, (Supplied with this documentation).
   This folder contains:
   a. The 'stand_alone' Folder with the following files:
      I. CMUcam2GUI_fe.jar, this is the Graphical User Interface, (GUI), program used to test and calibrate the camera in this process.
      II. Camera_Calibration.pdf, this file contains the instructions for the GUI.
      III. config.txt, this file contains the preset color parameters used in the GUI.
      IV. sserial.dll, this is the driver file for the GUI.

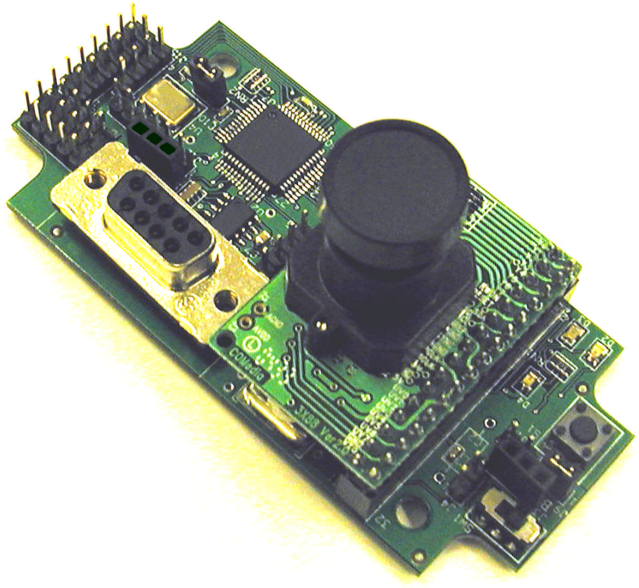
   b. The 'src' Folder that contains the source files for the GUI.

Contents

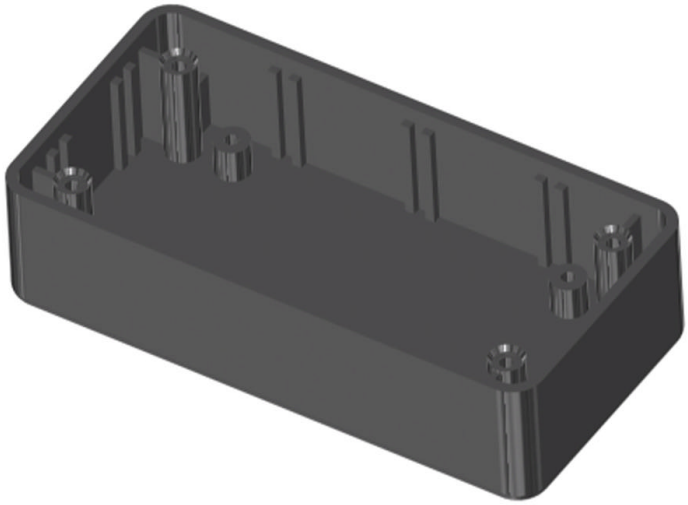
Required Items

This section identifies the items required to complete the process in this document.

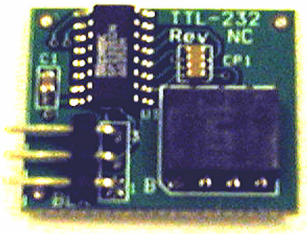
1. Camera assembly, (module with control board), from kit.



2. Camera enclosure, Radio Shack #270-1802. Available from Radio Shack.



3. TTL-232 adapter, (Robot Controller serial communications adapter), from kit.



4. (2) PWM cables from kit.

5. Hitec HS-322 Servo Motor from kit.

6. CMUCam2_fe Manual.

7. Computer with the following:
    a. Java Runtime Environment 1.4 or above installed, (Download from; http://java.sun.com/j2se/1.5.0/download.jsp)
    b. CMUcam2GUI_fe folder.
    c. MPLAB C18 compiler installed.
    d. IFILoader installed.
    e. Robot Controller programming cable.
    f. Robot Controller default code with camera code, FrcCode2005v2.4.

8. Robovation Kit parts, (see Camera Mount Assembly diagram).

9. Additional hardware:
    a. (4ea.) 4-40x1/4" Phillips head screw.
    b. (1ea.) 4-40x3/8" Phillips head screw.
    c. (2ea.) 4-40 nut.
    d. (4ea.) #4 flat washer.

10. ¼" plywood, (1-4'x 8' sheet).

11. Paint colors for targets as follows, (1 or more). These color mixes are from Home Depot. 1 quart is enough to paint up to 4 targets, two coats per target.

Blue Target

Behr Deep Base No. 1300			
COLORANT	**OZ**	**48**	**96**
E THALO BLUE	2	34	0
KX WHITE	0	13	0
V MAGENTA	0	0	1

Red Target

Behr Deep Base No. 1300			
COLORANT	**OZ**	**48**	**96**
KX WHITE	0	4	0
R EXTERIOR RED	1	37	0
V MAGENTA	1	6	1

Yellow Target

Premium Plus Interior Flat Wall Paint
SUNNY SUMMER (S-G-380)

Green Target, (Spray Paint)

RUST-OLEUM (Flourescent Bright Neon Colors)
1932 FLOURESCENT GREEN

White Primer

Behr BEHR 1-COAT PAINT
Water-Base Primer Sealer No. B1-2

Modification and Assembly

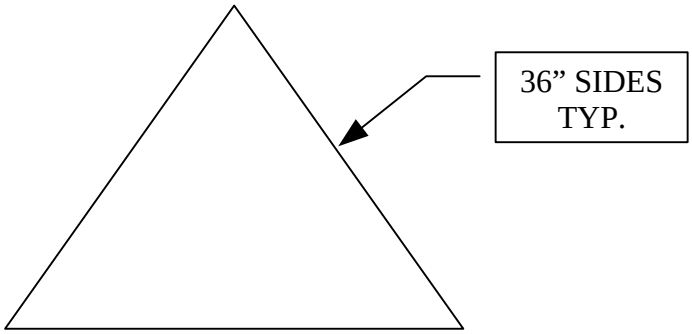
This section illustrates the modification and assembly of the components required in this process.

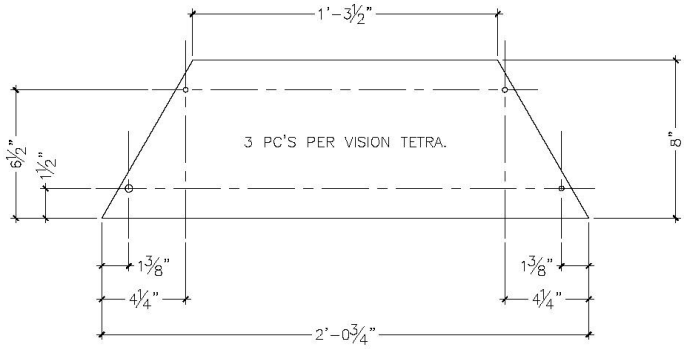
Camera Target

This section details the targets used in the goals, at the field borders, and on the vision tetra.

1. The targets will be made from ¼" plywood.

2. The target should be primed both sides and painted each side with a target color.
   a. The triangle targets used in the competition are YELLOW, RED, BLUE.
   b. The GREEN target is the Vision Tetra target.

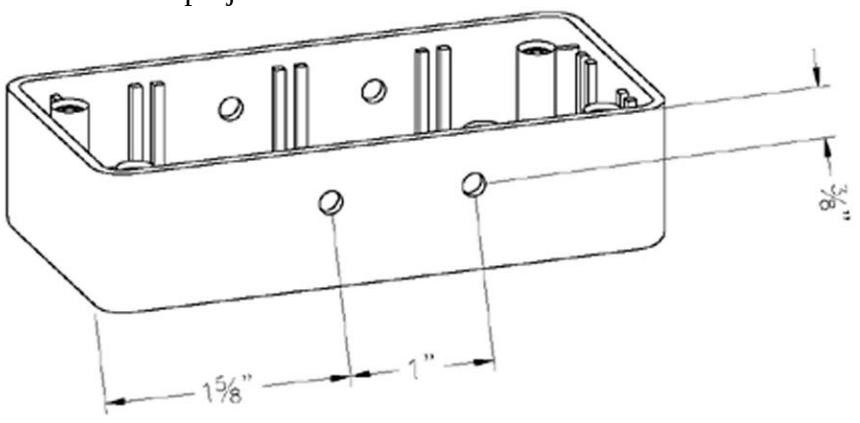




Project Box Modification

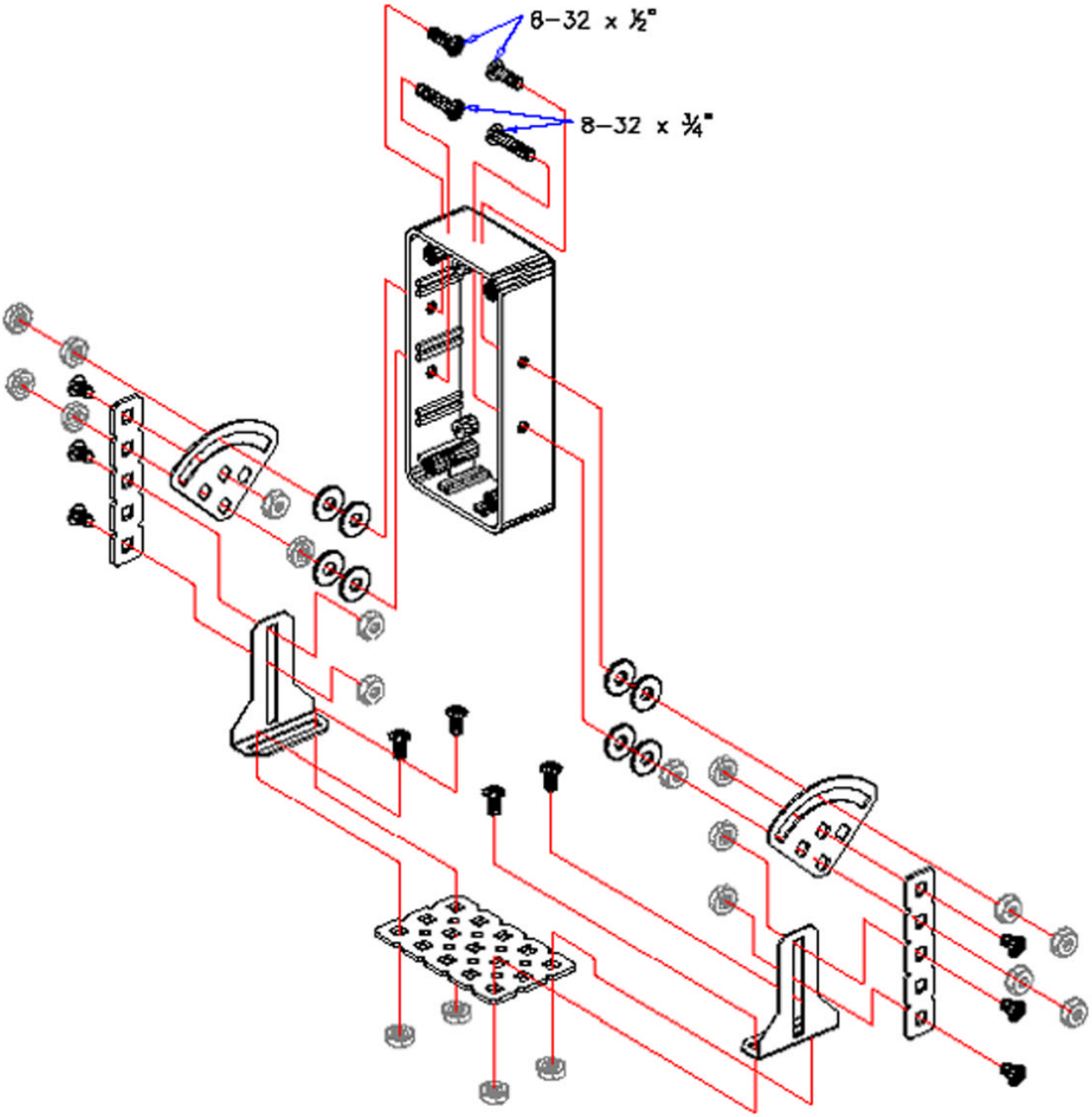
Detailed here is the modification to the Camera enclosure for attaching it to the Mount assembly.

1. Drill four .166" holes using the dimensions shown above. Be sure to measure from the same end of the box for each side of project box.

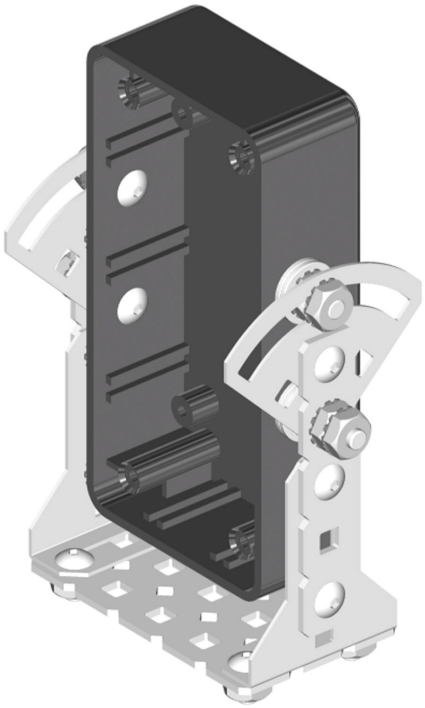


Camera Mount Assembly

This section illustrates the assembly of the Camera mount assembly. The assembly is constructed from parts in the Robovation kit plus the Radio Shack project box.



Fully assembled Camera Mount Assembly



Servomotor Coupler and Mount Assembly

This section details the modification of the servomotor coupler and the assembly of the coupler to the mount assembly / mount assembly to the servomotor. Part of this procedure requires connecting power to the camera assembly to power the servomotor. Care should be taken protect the camera assembly.

The Servo coupler

1. Drill the two outside holes of longest dimension .109" thru.
2. File off the raised edge around center hole to achieve a flat mounting surface.
3. Mount the coupler to and centered on the mount assembly using two each of the 4-40x1/4" screws, 4-40 nuts, and #4 flat washers. The center and enlarged holes of the coupler will align, side-to-side, on the three larger square cutouts through and centered on the bottom plate of the mount assembly.

Attaching the mount assembly to the Servo

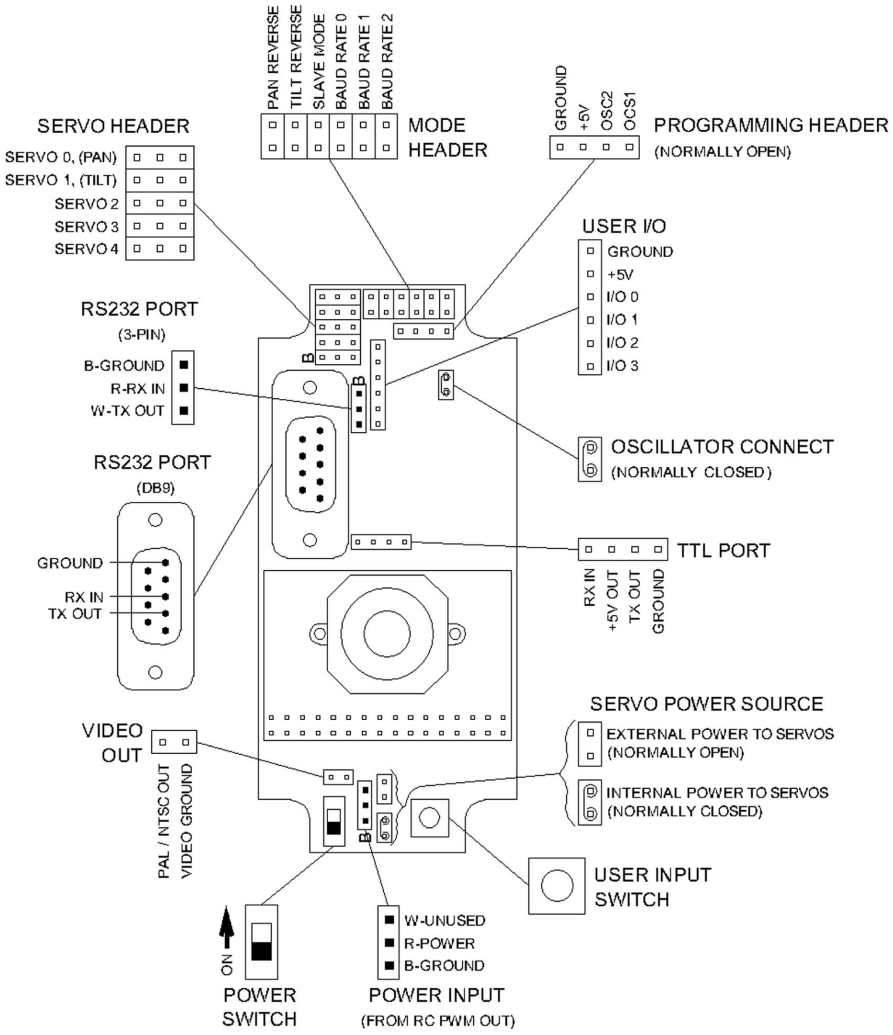
1. Connect the servo to the SERVO 0, (pan), connection and the power to the POWER INPUT connection on the camera control board per directions under connections on page 8.
2. Turn on the power to the camera control board. *The Servomotor should rotate the Camera / mount assembly to its 'home' position.*
3. Attach the mount assembly in desired position on the servomotor. Disconnect power from the camera control board. Secure mount assembly to servo with 4-40 x 3/8" screw and two #4 flat washers.

Mounting Camera assembly in enclosure

1. Mount the camera assembly into the enclosure with the power switch located at the end of the enclosure that will be closest to the floor when it is mounted on the Robot.
2. Secure camera to enclosure using two 4-40x1/4” screws in the holes provided.
3. Tighten only until screws and camera assembly are snugly attached, do not over tighten the screws.

Connections

This section shows the available connections on the Camera assembly and describes the connections used in this process.



Connections:

1. The <u>POWER INPUT</u> connection on the camera control board has a ‘B’ printed on the circuit board below the connector. This is the end of the connector where the black wire of the power PWM cable is connected. The female end of this cable will connect directly to any unused PWM Output of the Robot Controller, which will provide the power for the camera assembly. **NOTE:** ***The camera will not operate without the Backup Battery to the RC connected and fully charged.***
2. The <u>RS232 PORT, (3-pin)</u> data connection on the camera control board has a ‘B’ printed on the circuit board above the connector. This is the end of the connector where the black wire of the power PWM cable is connected. The female end of the cable will connect directly to the TTL-232 adapter.
3. Note the <u>SERVO HEADER</u> bank of connectors has a ‘B’ printed on the circuit board to the left side of the header. This is where the black wire of the Servomotor connector is connected.

Installation

Installation of the Camera, Mount, and Servo assembly:

1. Mount the Camera, Mount, and Servo assembly to the robot. The camera may be mounted at any height on your Robot with an unobstructed view of the area in front of the Robot. ***For this process only****, it is recommended the camera be located to place the lens of the camera thirty-six inches from the floor.* The Camera, Mount, and Servo assembly may be mounted using four stand-offs from the Robovation kit to connect the flanges of the Servomotor to a flat plate or by using the flanges on the Servomotor mounted to a bracket of your own design.

2. Run the PWM cables from the camera assembly to the Robot Controller, (RC). *Be sure to avoid running the cables where they are at risk of being damaged by moving components of your robot.* Leave enough cable at the Camera Assembly to allow the Camera / Mount Assembly to rotate without interference from the cables. *Since the camera can be sensitive to voltage fluctuations it is recommended that to extend a PWM cable, splice and solder two PWM cables together. Be sure to properly insulate the spliced wires.*

3. With the robot power off:
    a. Connect the PWM cable attached to the POWER connection of the Camera board to an unused PWM OUT port observing the position of the black wire of the PWM cable in relation to the label on the RC.
    b. Connect the PWM cable connected to the RS232 PORT, (3-pin) connection of the Camera to the TTL-232 Adapter observing the position of the black wire of the PWM cable in relation to the label on the TTL-232 Adapter printed circuit board. Lay the adapter on top of the RC for the time being. This will connect to the TTL data port on the RC after initial camera testing is complete.
    c. Verify the servomotor is still connected to the SERVO 0, (pan) connection of the Camera Assembly per the instructions under ‘Connections’ on the page 8.

Testing

The following steps require your Robot controls are wired and ready to operate. Also required is a fully charged backup battery appropriately connected to the RC.

Testing the Camera, Mount, and Servo assembly

1. With the computer, (see Page 3. Item 7.), booted up and ready to go:
    a. Connect the RC programming cable to the Serial Communication Port, (com port), of the computer and the RS232 PORT, (DB9) of the Camera Assembly. *The settings for the computer's com port should be:*
        - *Baud Rate: 115200*
        - *Data Bits: 8*
        - *Parity: none*
        - *Stop Bits: 1*
        - *Flow Control: none*.

    b. Place the power switch on the Camera assembly into the on position, (switch positioned closest to the Camera lens.)
    c. Apply power to the Robot. *The RED and GREEN LEDs on the Camera printed circuit should light and the Servomotor should rotate the Camera / mount assembly to its 'home' position*.

2. Testing the Camera assembly using the CMUcam2GUI_fe
   Open the folder labeled 'CMUcam2GUI_fe/stand alone' and double click on the program titled CMUcam2GUI_fe.jar. *A small pop-up window will open at the upper right on the screen labeled 'Serial Port Select'*. Enter the number, (1, 2, 3, or 4), of the computer's com port you are using in the text box of the pop-up window and click OK. *The program window for the CMUcam2 GUI should open and after a brief moment "CMUcam Version 2 type 6 ready." Should appear in the text box labeled 'Console:'* Maximize this window. *If the GUI does not appear to be working, refer to page 13 of the CMUcam2_fe Manual.*
    a. With the lens cover removed from the camera, click on the 'Grab Frame' button at the bottom of the CMUcam2GUI_fe, (GUI). *The status bar below the 'Grab Frame' button should start to fill from the left and the RED LED on the camera board should flicker until the image is loaded, (visible in the GUI window).*
    b. To focus the image:
        I. Turn the outer most part of the lens up to one-half turn in either direction, (clockwise, counter-clockwise), noting the direction.
        II. Acquire a new image using the 'Grab Frame' button.
        Inspect the image on the screen. If it is less focused reverse the direction you turn the lens and repeat from step 'I.' until the image is focused. *Although the image is low resolution, you can identify good focus by looking at sharp edges, and you will find a focus ring position for which everything is in focus except for very close objects.*

Calibration

This section will guide you through the process of using the GUI to acquire the Automatic Exposure Control, (AEC), settings used in the vision code for your Robot. From this calibration process you will obtain three AEC settings, (for Yellow, Green, Red). These three AEC settings allow the vision code to track on five colors preprogrammed into the vision code. The five color availability is due to the Yellow AEC setting also applies to White and the Green AEC setting also applies to Blue.

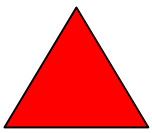
Calibrating the camera to the environment.

1. The calibration of the camera to the environment will be performed using the vision system you have just assembled, painted targets, (Green, Yellow, and Red), and the imported sections of the Camera_Calibration.Pdf that follow.
   NOTE: The calibration procedure will have to be performed at each location you wish to use the Camera.

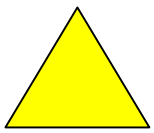
Camera_Calibration.Pdf

Step 1: Setup and Position the Camera

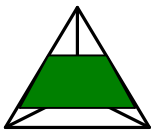
In order to be able to track all five of the FIRST specified colors, you must calibrate on three different targets shown below. The order in which you calibrate does not matter. It is best to train the target in the position it will be used on the field. If for some reason you find that your normal light is not bright enough (this can occur with certain florescent lights), consider illuminating just the target with a more concentrated external light. In general, yellow and green tend to be more robust even in lower light conditions.



Flat Red Triangle



Flat Yellow Triangle

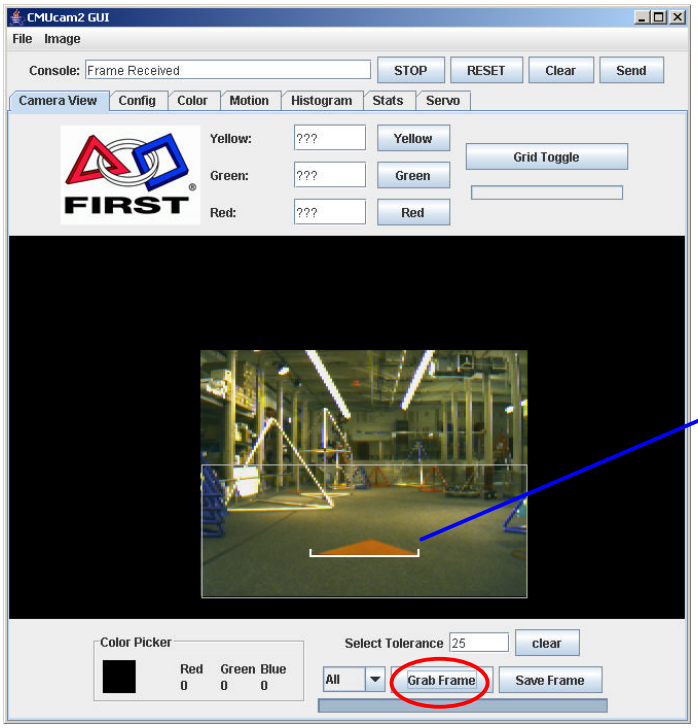


Green Targeted Vision Tetra

1. Make sure the camera is 11 ft from the target.
2. Make sure the camera is 36 inches from the floor.
   - You can use two stacked tetras as a stand.
3. Using a piece of guide rope can help in the alignment.
4. Point the camera toward the target. Keeping the lens body parallel to the floor.

Step 2: Starting the CMUcam2_fe GUI
Omitted section

Step 3: Grab a Frame and Align the Target

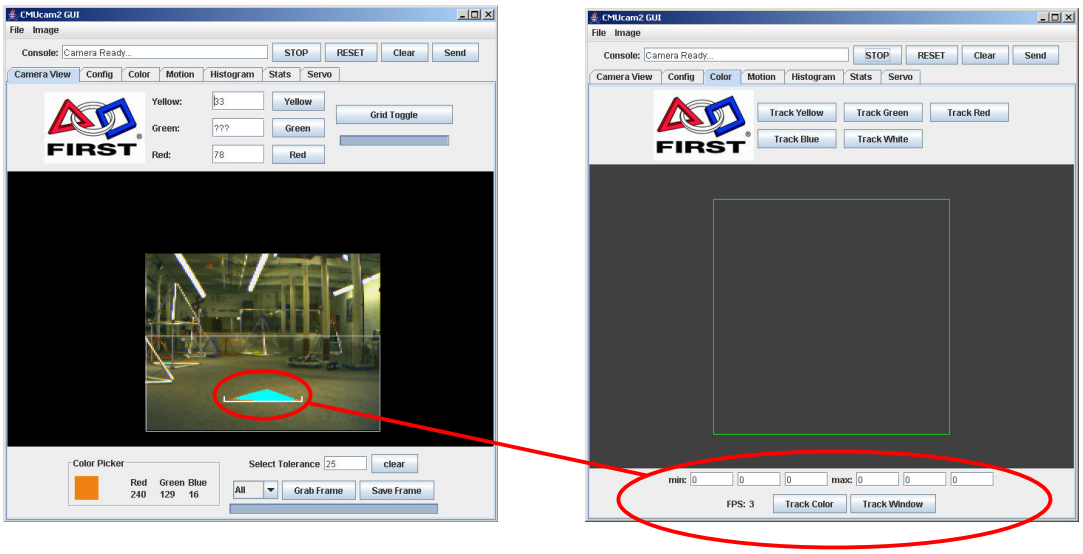


Next, **Click** the "**Grab Frame**" button (circled in red) to see where your target is located. After about 30 seconds, an image should appear in the middle of the screen.

Try to **center the target** as best as possible inside the "Alignment Bracket". You may need to tilt the camera slightly to correct for any vertical error. Grab a few more frames to be sure.

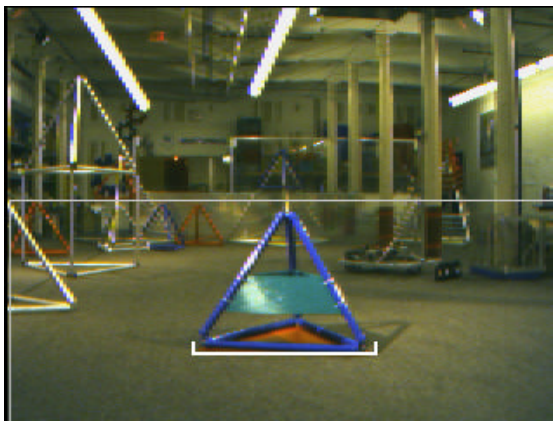
Also, make sure that the area inside the light grey calibration bounding box is clear of colorful objects such as other tetras or targets. This is the region of the image that the camera will use for its calibration.

One useful feature to explore is the ability to scroll the mouse pointer over the image to see the Red, Green, Blue, (RGB), color values for each point in the image. These values are displayed under their titles next to the Color Picker in the Camera View window. When you click on a color, all occurrences of that color in the image are highlighted, and the colors Red, Green, Blue, (RGB), values are filled into the min. and max. boxes of the "Color" window. The min. and max. values are determined by adding / subtracting the value in the Select Tolerance text box to / from the mean values of the color the cursor was on when you clicked on it.

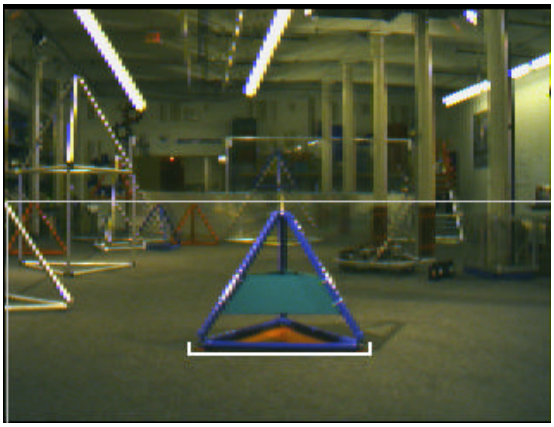


Step 4: Calibrating

Before calibrating the target "grab" a frame and inspect it closely for instances of glare. Shown below are two pictures; the one on the left has an excessive amount of glare, the one on the right was taken after the target was slightly rotated.

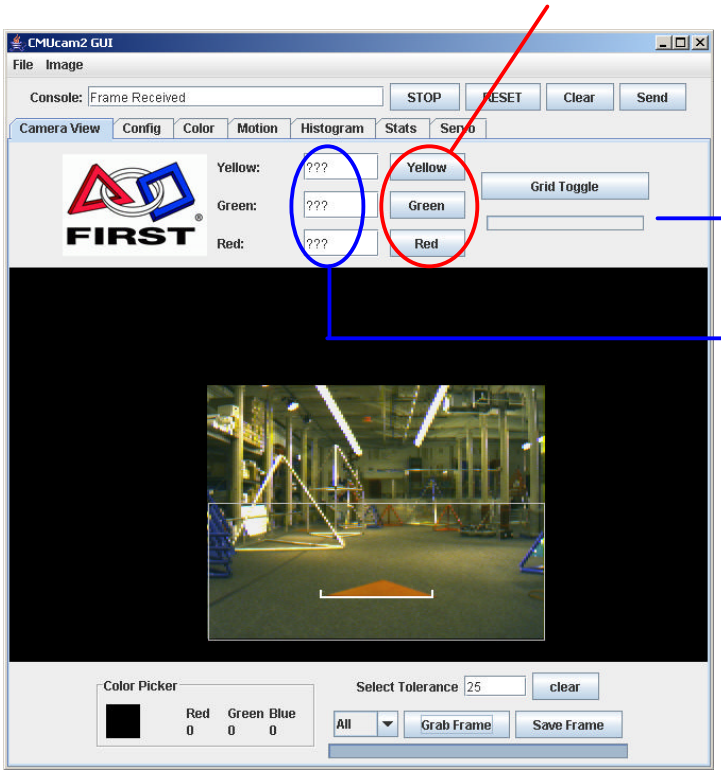


A Bad amount of glare



Target slightly rotated to remove glare

Next, **Click** on the name of the **color** target that you have centered in the screen.

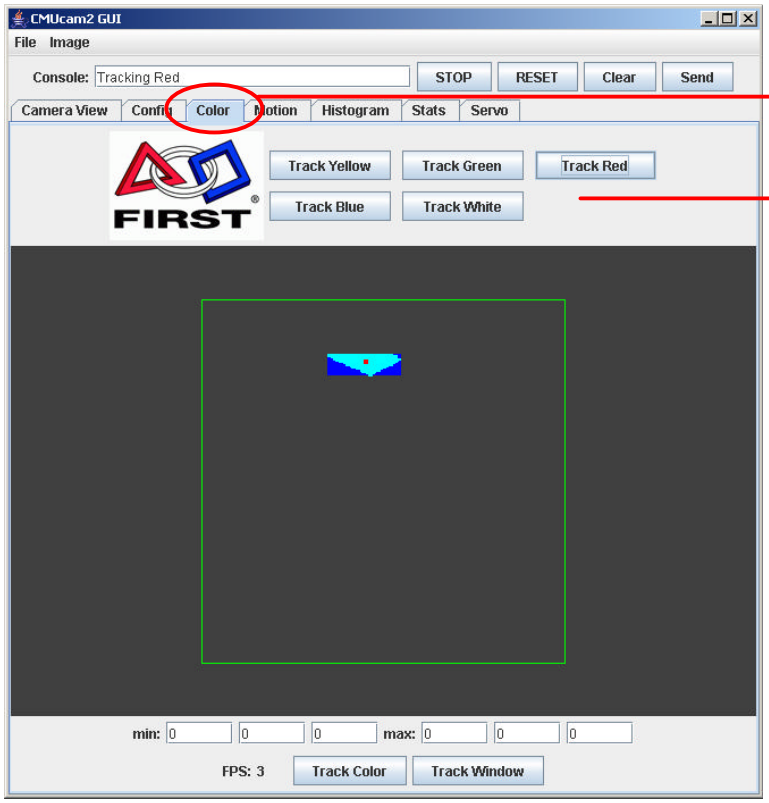


The progress bar below "Grid Toggle" should start moving as the camera is calibrating.

Upon successful completion, the "???" in the text box next to the color's name should be replaced with the **correct exposure value**.

Step 5: Testing the Calibration

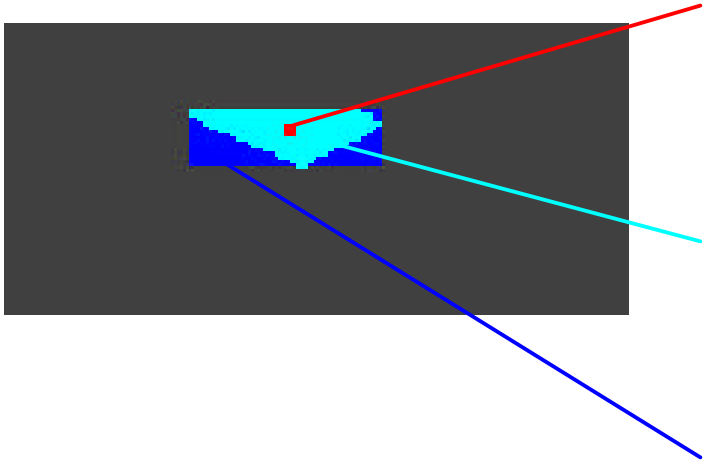
Once you have calibrated each color, you can test to see how well the calibration is working.



Click on the "**Color**" **tab** to switch to the color tracking panel.

Then **Click** on the **color** that you wish to track. For example, here the red target is being tracked.

*Note that if you manually enter the exposure values on the previous panel, and then you switch back to the "color" panel, your new manually entered values will be used.



The Red Dot shows the "centroid" also referred to as the middle of mass of the image. This is where the camera has calculated the center of the object is located. The middle of mass is what you should use when trying to drive towards or follow an object.

The light blue color shows a low-resolution image of the exact regions in the image the camera detects are close to the color you are tracking. Here we can see our red triangle from before. This feature of the CMUcam2 is called "linemode".

The dark blue box shows the bounding box that contains all tracked pixels. It is the smallest rectangle that is parallel to the x and y axis that holds all of the tracked sections of the image. If this dark blue area fits tightly around the cyan colored blob, then you have a great track on the object.

How Calibration Works

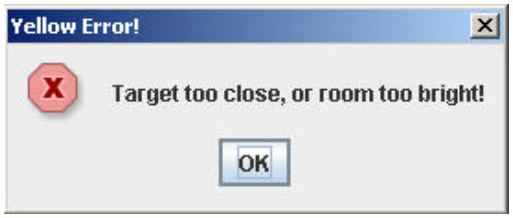
The camera calibration program is given the approximate size, (in pixels), and location, (distance), of the target object. It uses this information in the process it uses to "pick" an Automatic Exposure Control, (AEC), setting. It starts off initially with the AEC set to 0. At this level, the image should be nearly black. It then uses hard coded tracking parameters for the color and checks to see if it can find the object. If it does not find the object, it increases the AEC setting and then tries again. As the AEC setting starts to get closer to a good value for the scene, the object will start to be found by the camera. The program stops the search when it can track an object that is large enough to be the target that you placed in front of it and reports the AEC setting.

Troubleshooting

The CMUcam GUI_fe will not start, or Java does not appear to work.

If this turns out to be the case, please refer to the CMUcam2_fe_manual Documentation provided by FIRST. It has a more detailed explanation of how to install and setup the camera.

The following error dialog pops-up when I try to calibrate on a color:



This error message occurs when the target fills up too much of the camera's view. This could be because the target is too close, too bright, or that the background does not have a large enough contrast with the target. Make sure that you have a dark carpet / floor underneath where you are training.

The following error dialog pops-up when I try to calibrate on a color:

This can be caused if the target was not correctly place in the training brackets. It can also be caused by too little light. Make sure that you have at least 75 foot candles of illumination on the target you are trying to train.

None of the target colors specified will work when I try to calibrate on them:
This can happen if the lighting is insufficient for the camera to "see" the colors as they are defined for the FIRST competition field targets. If you are sure the camera is working they you should try;

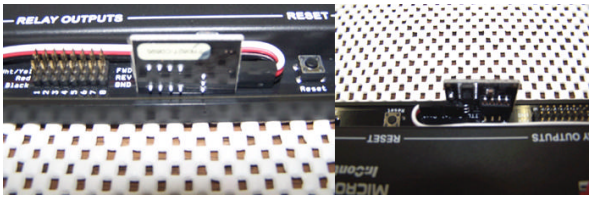
a. Increasing the lighting in the environment you are testing in.
b. Choosing a different target color.
c. Moving to another area to test the camera.

Loading and testing the Robot vision code

This section describes the process of loading the code and performing operational testing of the vision system on your Robot.

The code is setup to turn drive control over to the Camera using the data that represents the Pan Servo position the Camera generates to track a target. **<u>This code requires you use PWM outputs 11 and 12 on the RC to control your drive motors</u>.** For testing and calibration purposes the trigger switch of the joystick connected to PORT 1 of the OI controls the activation of the vision code.

1. To enter the calibration parameters into the Robot vision code:
    a. Using the MPLAB IDE compiler open the project ”frcCode.mcp”.
    b. From the project window under ‘source files’ open the user_routines.c file.
    c. Scroll down to the heading ‘INITIALIZE CAMERA’.
    d. Find the line ‘camera_init(64,85,50); // Set the 3 exposure values from calibration GUI’. (From left to right these three numbers are the Automatic Exposure Control, (AEC), settings for YELLOW, GREEN, RED.).
    e. Enter the calibration value for the color(s) in the appropriate position.
    f. Compile the code.
2. Using the Innovation First IFI Loader program download the code to the RC.
3. Connect the TTL-232 adapter to the RC 4-pin TTL port. The TTL-232 adapter has ‘BLK’ printed on the printed circuit board. This is the end of the connector where the black wire of the power PWM cable is oriented. The TTL-232 adapter plugs directly to TTL port of the Robot Controller.

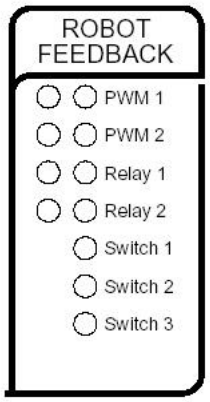


Setting the trim of the joystick

1. Be sure the drive motors are controlled by PWM outputs 11 and 12 on the RC.
2. Place the Robot on blocks to allow the drive wheels free to turn without moving the Robot.
3. Apply power to the Robot and the Operator Interface, (OI), and verify radio communication.
4. This code is written to utilize single Joystick control over the robot and requires the joystick be “Trimmed”, (balanced).
    a. Set the trim for the Joystick connected to port #1 of the OI:
        I. Slowly turn the vertically oriented thumbwheel at the right of the joystick until one or both drive wheels of the robot spin slowly.
        II. Then slowly turn the horizontally oriented thumbwheel until both drive wheels spin at the same speed.
        III. Slowly turn the vertically oriented thumbwheel at the right of the joystick until one or both of the drive wheels of the robot stop.
        IV. Repeat step II & III as necessary until the wheels no longer spin without operation of the Joystick.

Selecting a color to track

1. To select the color you want to track you can toggle through the color options by squeezing the trigger on the joystick connected to Port 2 of the OI. The right LED will light GREEN to indicate your choice of color as shown below.



Yellow
Green
White (not used in the 2005 FIRST competition.)
Red
Blue

   a. As the color is selected using the joystick trigger the camera will attempt to track on the color currently selected. *When camera tracks on a color a red LED will light on the CMUcam2 itself will light whenever the camera is successfully detecting the color target. This is a useful diagnostic tool so that you can tell whether, at the most basic level, the CMUcam2 is "seeing" the target or not. If CMUcam2 does not see the target, then everything downstream of the camera is not at fault, and it is time to think about lighting and calibration.*
   b. While squeezing the trigger of joystick #1, have someone move the target object, whose color you entered into the code, across the path of the Camera. *The Camera should swivel, (Pan), as it tracks the object and the drive wheels should behave as if you were steering the Robot with the Joystick.*
   c. Place the Robot on the floor ten to fifteen feet away from the target at an angle to the target severe enough to cause the Robot to have to turn but not so severe that the Camera will not see the target.
   d. While squeezing the trigger of joystick #1, observe the robot and how it steers to the target. If the robot's forward speed is too slow or fast it can be changed as follows:
      i. Remove power from the OI.
      ii. Move the Robot within reach of the RC programming cable.
      iii. Connect the RC programming cable to the 'Program' port on the RC. The MPLAB IDE compiler should still be running with the project "FrcCode.mcp" open and the file 'user_routines.c' still in view.
      iv. Scroll to the heading 'VISION VARIABLES'.
          Find the constant < const int speed_setting = 0; >.
          Increase this variable by 10 to increase the forward speed
          Decrease this variable by 10 to decrease the forward speed
          The speed_setting value must be a positive number and must not exceed 127.
      v. Compile the program and download the code to the RC.
      vi. Disconnect the RC programming cable from the RC.
      vii. Connect power to the IO. Perform Steps 4b. 4c. and 4d. of this paragraph until the Robot steers directly to target.

e. If the Robot does not steer directly to the target then steering compensation must be used.
   i. Remove power from the OI.
   ii. Move the Robot within reach of the RC programming cable.
   iii. Connect the RC programming cable to the 'Program' port on the RC. The MPLAB IDE compiler should still be running with the project "FrcCode.mcp" open and the file 'user_routines.c' still in view.
   iv. Scroll to the heading 'VISION VARIABLES'.
      Find the constant < const int steering_comp = 0; >**.**
      Increase this variable by 5. *If it is necessary to set steering_comp to a value greater than 70 you may want to modify your Robot to improve it's steering ability. One way to do this is to shift the weight on the Robot, or add weight, to the area where the drive wheels are located.*
   v. Compile the program and download the code to the RC.
   vi. Disconnect the RC programming cable from the RC.
   vii. Connect power to the IO. Perform Steps 4b. 4c. and 4d. of this paragraph until the Robot steers directly to target.

Congratulations!!! You have reached the end of the **Getting Started** process.

At this point you have successfully Assembled, Calibrated, and Tested a vision system as part of your Robot. A vision system that, as-is, can track on any one of up to five colors and drive your Robot to that color.
It is envisioned that many of the Robot designs will use touch or proximity sensors to judge when it reaches a Tetra, Goal, or Loading Station. Some may use the Accelerometer or other more complex sensors to achieve the same results.

From this point it is up to you to take the vision system and what you have learned to the next level… to provide your Robot with the skills it requires to perform the tasks intended for it

The Code

CMUcam2 FIRST Robot Controller Application Programming Interface

Introduction

This document contains information about how to interface the Robot Controller, (RC), with the CMUcam2. The CMUcam2 is an embedded color vision sensor that can give your robot the ability to "see" colored targets. The CMUcam2 has its own set of low-level functionality that goes beyond the scope of this document, but can be found in the CMUcam2 Manual. This document discusses the different functionality that is immediately accessible using the Application Programming Interface (API) provided by FIRST.

The main use of the CMUcam2 in the 2005 FIRST Robotics Competition, (FRC), will be to track colors. Tracking colors essentially means that the camera will tell you where a large concentration of a certain color can be found in the image. If you imagine taking a picture that is printed on graph paper, you can describe a location in the image with an X and a Y coordinate. The CMUcam2 uses this same type of coordinate system to tell your Robot where a colored object can be found. Your Robot can then use these values to direct itself towards a colored target. Figure 1 depicts a diagram showing what the coordinate system of the CMUcam2 looks like from the camera's perspective.

The FIRST API already contains a set of functions that you can use to track some predefined colors. It also shows samples of how to set some of the camera parameters. The following files are required in order to use the CMUcam API:

```
user_camera.c
user_camera.h
user_SerialDrv.c
```

The bulk of the code can be found in "user_camera.c". The "user_camera.h" file contains the function prototypes and a few definitions used by "user_camera.c". The "user_SerialDrv.c" file contains the communication interface for communication between the robot controller and the camera.

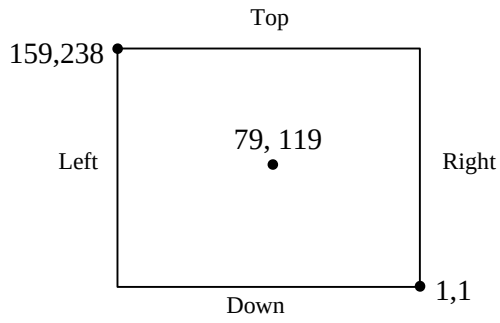


Figure 1

This figure shows how the coordinate system is oriented with respect to the robot. If the target is being tracked at 79 across and 119 up, that means it is in the middle of your field of view. If the x (horizontal) value is smaller than 79, then the robot would need to turn right to center it. If the object has an x value greater than 79, it is to the left of the robot. The same is true for the y (vertical) axis. Greater than 119 means it is up, and less than 119 means it is down. A value of 0, means that the object is not within view of the camera.

Function Descriptions

`int camera_init(int exp_yellow, int exp_green, int exp_red)`

The `camera_init()` function initializes the camera. It programs the camera with the tracking exposure levels for your particular environment. Different lighting conditions can drastically effect how well the camera can track colored targets. Even though people are easily able to tell the difference between yellow in bright light, and yellow in a shadow, to a camera these colors may be quite different. One way to help reduce the effects of lighting levels, is to adjust the cameras Automatic Exposure Control, (AEC). The AEC controls how long the camera will wait for light before it starts collecting the next image. The higher the exposure value, the longer the camera waits, and hence the brighter the image looks. The `camera_init()` function takes three different exposure values as arguments, each for a different colored calibration target. These three exposure values are used by other colors, so it is important to set them even if you are not planning on using yellow, green or red, (although yellow and green are the typically very good colors). When you go to a new environment (such as a FIRST event), you should update your exposure values so as to capture the new lighting conditions. This function returns 1 on success and 0 if a failure is encountered. Failures may be due to loose cables, low power, or anything else that might disturb the camera's operation.

Here is an example that initializes the camera with yellow, green and red exposure settings equal to 22, 54, and 30 respectively:

```
success = camera_init( 22, 54, 30 );
```

You should put a command like this inside `User_Initialization()` in your "user_routines.c" file after you call `Serial_Driver_Initialization()`.

`int camera_find_color(int color);`

The `camera_find_color()` function programs the camera with the parameters necessary to track one of the following colors. It then sends the camera the command to begin tracking, "TC":

```
YELLOW
GREEN
WHITE
RED
BLUE
```

For example, to start tracking the color yellow, you would write:

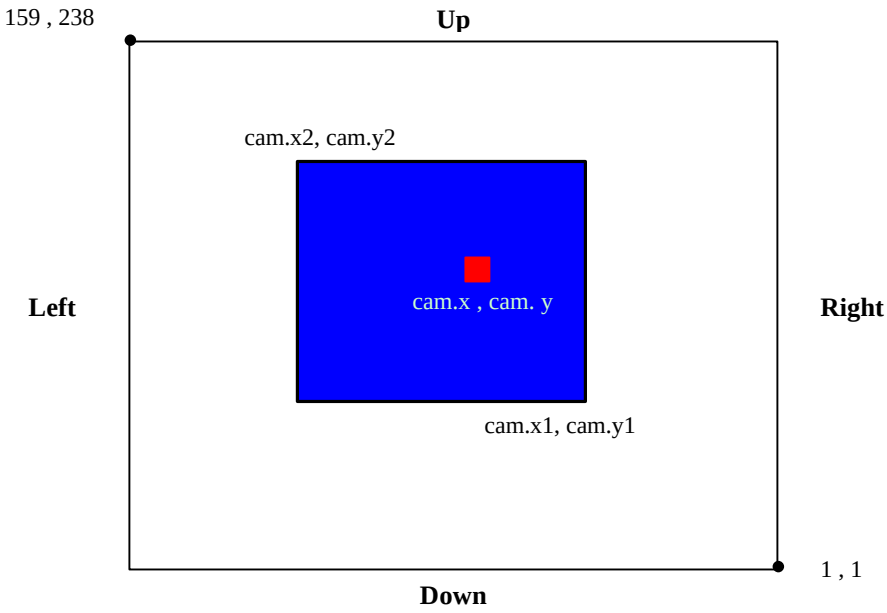
```
camera_find_color( YELLOW );
```

This function returns 1 on success and 0 if a failure is encountered. After calling this function, you will need to call `camera_track_update()` before updated tracking values will be available to the RC. To stop a tracking command, use the `camera_stop()` function. **You will need to stop the camera before initiating any other camera commands, except for `camera_track_update()`.**

```
int camera_track_update(void);
```

The camera_track_update( ) function tells you when a new packet is ready for you to use. If the packet is ready, the function will return 1, and update the global *cam* data structure. The different elements of the *cam* data structure are listed below and can be accessed anywhere in your code.

NAME	DATA TYPE	DESCRIPTION
cam.x	unsigned int	The x coordinate of the middle of mass (Red Square)
cam.y	unsigned int	The y coordinate of the middle of mass (Red Square)
cam.x1	unsigned int	The lower right horizontal value of the bounding box (Blue Rectangle)
cam.y1	unsigned int	The lower right vertical value of the bounding box (Blue Rectangle)
cam.x2	unsigned int	The upper left horizontal value of the bounding box (Blue Rectangle)
cam.y2	unsigned int	The upper left vertical value of the bounding box (Blue Rectangle)
cam.size	unsigned int	The number of pixels tracked by the camera
cam.conf	unsigned int	The confidence of the camera track
cam.pan_servo	unsigned int	The pan value of the servo
cam.tilt_servo	unsigned int	The tilt value of the servo



Since the camera may process frames slower than your main control loop, you need to check to see if new data is ready before you use the *cam* structure. This is easy to do with the return value from camera_track_update(). If the return value is 0, then repeat your last action, or do nothing. If the return value is 1, then process the values from the new structure.

Below is in example of how you could access the different elements of the camera data when they are ready in “user_routines.c”:

```
// The camera library is included
#include “user_camera.h”
#include “user_Serialdrv.h”
…
// Somewhere at the top, the global cam array is defines
extern cam_struct cam;
int counter=0;
…
void User_Initialization (void)
{
…

  Serial_Driver_Initialize();
  Putdata(&txdata);             /* DO NOT CHANGE! */

/**********************
** INITIALIZE CAMERA **
**********************/
camera_init(64,85,50); // Set the 3 exposure values from calibration GUI

}


void Process_Data_From_Master_uP( void )
{
int I,x,y;
  Getdata(&rxdata);   /* Get fresh data from the master microprocessor. */

  if (PC_Is_Not_In_Control()) Default_Routine();

if(counter==0) // First time the function is called
      {
      i=camera_find_color(GREEN);
      counter=1;
      }
else          // Every time later, we run this part of the code
      {
      if( camera_track_update() == 1 )  // Check to see if the camera is ready
            {
            // You got a new color track packet
            x=cam.x;
            y=cam.y;
            if(x==0 || y== 0 ) break; // It doesn’t see anything :(
            // Do something with the new x and y value
            // Yes, we can drive the robot based on x and y now…
            }
      }

}
```

`int camera_stop(void);`

The `camera_stop()` function is used to stop the camera's current processing.
This function sends a '\r' command to the camera to stop it from streaming data and then checks to make sure that it sent an ACK back. 1 is returned upon success, 0 on failure.

`int camera_auto_servo(int enable);`

This function enables the camera's internal auto servo mode. A value of 1 enables it, while a value of 0 disables it. When auto servo mode is enabled, the camera will output a pwm signal from its pan and tilt servo ports that try to point the camera towards the color being tracked. It will also update the *cam.pan_servo* and *cam.tilt_servo* positions in the global camera structure. On success it returns 1, and on error a 0. Note that this command will interrupt tracking, so you should send `camera_stop()` before using this command.

Here is an example of how to turn on auto servo mode and check for an error in sending the command:

```
camera_stop()
val = camera_auto_servo( 1 );
if( val== 1 ) printf( "Success!\r" );
else printf( "Doh, that failed…\r" );
```

`int camera_set_servos( int pan, int tilt );`

If you are using the servo ports on the camera, this function can be used to manually set a pan or tilt position for the CMUcam2 servos. Sending servo commands using the camera will interrupt any on going tracking, and hence makes manual control quite slow. If you are tracking and wish to use this command, remember to send `camera_stop()` before sending a servo position. Both the pan and tilt values range from 46 to 210 with 128 as the middle value. It returns 1 upon success and 0 upon failure.

Here is an example of how to set the servo positions to the middle:

```
camera_stop()
val = camera_set_servos( 128, 128 );
if( val== 1 ) printf( "Success!\r" );
else printf( "Doh, that failed…\r" );
```

Constants and Defines

user_camera.h

```
#define MAX_BUF_SIZE 64       // Maximum size of the serial buffer
#define CAMERA_PORT 1         // Picks which UART to use when talking to the
                              // camera
#define FREEZE_ON_ERROR 1     // Causes the detection of an error to halt
                              // the robot with a program state error.
                              // Set to 0 for competitions.

// Color Definitions for the track color command
#define YELLOW 1
#define GREEN 2
#define WHITE 3
#define RED 4
#define BLUE 5

// Different color spaces that the camera can operate in
// (see CMUcam2 Manual for more information about YCrCb color)
#define RGB   0
#define YCrCb 1
```

user_camera.c

At the top of user_camera.c, you can find this section of code that explains all of the different tracking parameters that work well for our five field colors. Some of them are in yCrCb color space, not in (RGB color space), so be very careful with what you change. You may be able to find better tracking settings with some experimentation. Just be sure they are robust to different lighting conditions. These can be experimented with quite easily using the java GUI.

```
// AEC - Automatic Exposure Control
// AGC - Automatic Gain Control
// RGB - Red Green Blue color space
// YCrCb - Another color space (not RGB), that is less effected by lighting changes


rom const char *noise_filter = "nf 6";         // Set noise filter to build up a tolerance
                                               // to stray pixels
rom const char *agc_level = "CR 0 32";         // Set AGC gain to mid level
rom const char *yCrCb_mode = "CR 18 0";        // Set into yCrCb mode instead of RGB
rom const char *track_green = "TC 85 120 0 255 80 150";
                                               // Track green color in yCrCb mode
                                               // Cr (85, 120)
                                               // Cb (80, 150)
                                               // Y  (0, 255)
rom const char *track_yellow = "TC 100 255 75 255 0 20";
                                               // Track yellow color in RGB mode
                                               // R (100, 255)
                                               // G (75, 255 )
                                               // B (0 , 20 )
rom const char *track_white = "TC 150 255 150 255 50 255";
                                               // Track yellow color in RGB mode
rom const char *track_red = "TC 190 255 0 255 0 40";
                                               // Track red color in YCrCb mode
rom const char *track_blue = "TC 0 150 0 150 70 255";
                                               // Track blue color in RGB mode
rom const char *aec_disable = "CR 41 128";     // disable automatic AEC settings
rom const char *manual_agc = "CR 19 32";       // allows manual setting of AGC
rom const char *raw_mode = "rm 1";             // set data mode to hex output
```