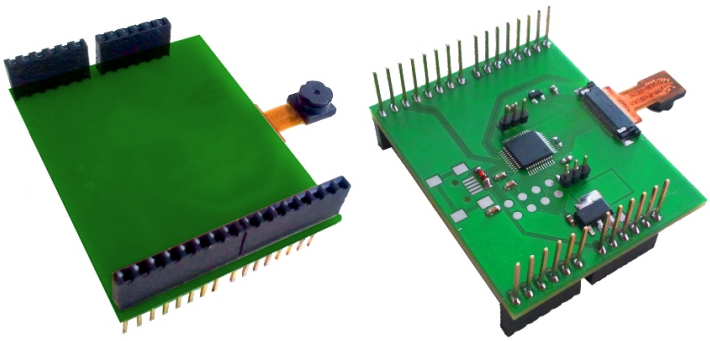
HaViMo2.5

Arduino Image Processing Shield

August 31, 2012



Features

- Integrated Color CMOS Camera
  - Frame Resolution: 160*120 Pixels
  - Color Depth: 12 bits YCrCb
  - Frame Rate: 20 Fps
  - Full Access to all CMOS Camera registers
    - Saving values in Flash, no need to reconfigure after power on
    - Auto / Manual Exposure, Gain and White balance
    - Adjustable Hue/Saturation
- Color-Based Image Processing
  - Integrated Color Look-up Table
    - Saved in FLASH, No need to recalibrate after power on
    - Up to 256 Objects can be defined
    - 3D viewing and editing tools
    - Real-time LUT overlay on the Camera Image
  - On-line Region-growing

        * Detection of up to 15 contiguous Regions per Frame
        * Reporting Color, CoM, Number of Pixels and Bounding box for each region
        * Adjustable Noise / small Region filtering
    - On-line Gridding
        * Reduces the Resolution of the Image to 32*24
        * Minimum Loss of Information using Object Priority
        * Reports Color and Number of Pixels for each 5x5 Cell
    - Raw image output in both calibration and implementation modes
        * Full Frame Output at 0.5 FPS
- Supported Hardware
    - Full Duplex
        * Arduino
        * Any microcontroller with UART
    - Requirements for generic microcontroller platforms
        * TTL Level 115200 BAUD Full Duplex UART
- Supported Software
    - HaViMoGUI

1 Introduction

HaViMo is a computer vision solution for low power microprocessors. It is equipped with a CMOS camera chip and a microcontroller which performs the image processing. The results are then accessed via serial port. In HaViMo2.5 several features such as frame rate are improved.

Region growing algorithm is capable of recognizing colored blobs in the image. It reports the bounding box as well as the centroid and number of pixels in the blob.

Gridding algorithm is a suitable pre-processing step for many other applications such as shape based object recognition and self localization.

HaViMo2.5 is shipped as an Arduino shield. It is compatible with all microcontroller platforms capable of establishing a UART connection. Communication with Arduino is performed via soft-serial port.

2 Hardware Setup

Figure 1 shows the pinut of the module.

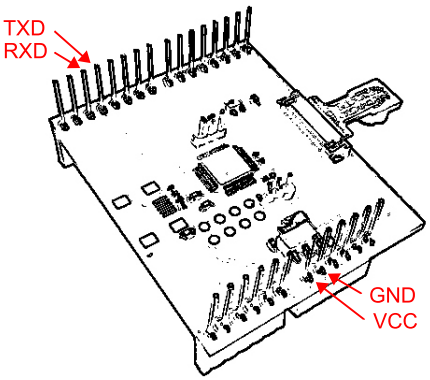
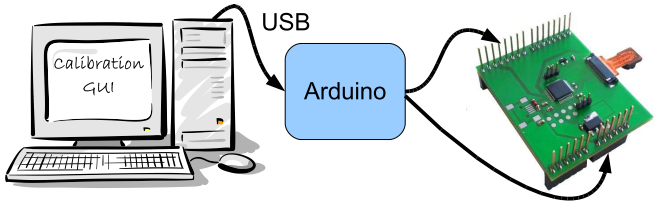
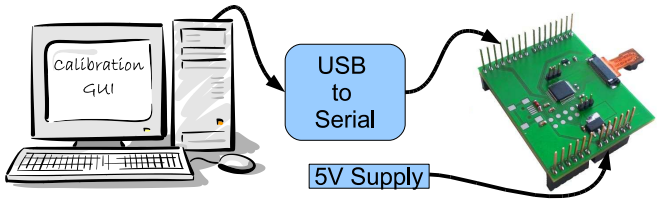


Figure 1. Pinout of the Module

The module can be used in different configurations according to the hardware platform it is used in conjunction with. These are described in calibration and implementation modes.



b) Direct Calibration



c) In-System Calibration using Arduino/Generic Microcintroller

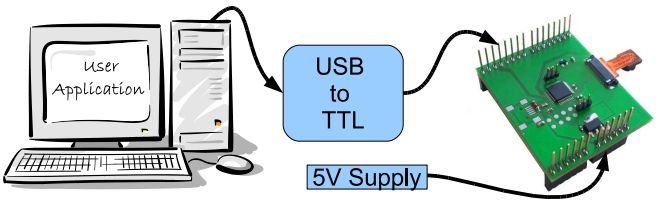
Figure 2. Possible Hardware Configurations in Calibration Mode.

In calibration mode the module is connected to a PC where a GUI facilitates the access to camera parameters as well as the color look-up table. in this mode, the camera chip can be configured and color to object associations are established by user according to the lighting conditions.

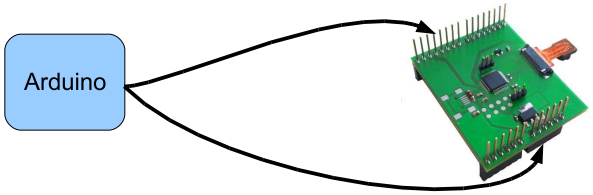
Possible configurations in calibration modes are:

- Direct Calibration using a USB to serial adapter
  - The module is connected through a USB to serial connector to a PC. Note that it is necessary to connect the power supply externally to the device.
- In-System Calibration with an Arduino

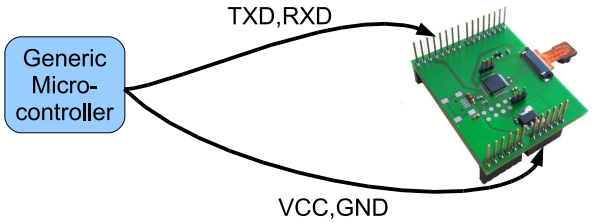
– The module is connected through an Arduino to a PC. A special program on Ardioono allows data communication between HaViMo2.5 and the PC



a) Implementation on a PC Platform using USB2Serail Converter



b) Implementation Mode with Arduino



c) Implementation Mode with a Generic Microcontroller

Figure 3. Possible Hardware Configurations in Implementation Mode

3 Function Description

HaViMo2.5 is accessed on a serial bus. The communication protocol for accessing the device works on a command/response basis. A command packet contains an instruction which invokes a function of the device, reads or writes values or a combinations of these. In this section the function of the device is described.

3.1 Communication Protocol

HaViMo2.5 supports full duplex communications in physical layer. This makes the module compatible with a wide range of microcontroller platforms. Following diagram shows a summary of command and response packets.

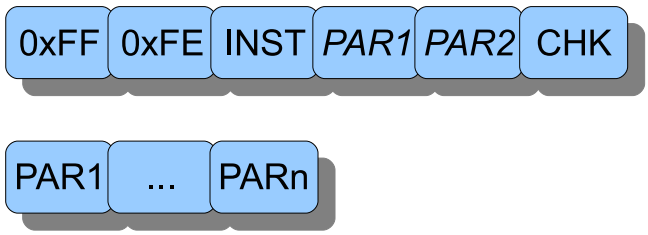


Figure 4. Command and Response Packets of HaViMo2.5.

The command packet is structured as follows.

Header The sequence 0xFF 0xFE

INST Instruction code described in table 1.

PAR1,PAR2 Parameters associated with the instruction. Note that The number of parameters MUST always be 2

CHK Check sum is the lowest 7 bits of the exclusive or of the instruction and the parameters.

3.2 Instructions

Following table shows available instructions in HaViMo2.5.

Instruction	Value	Params.	Function
PING	0x01	0,0	No action. Used for obtaining a Status Packet
READ_REGION	0x02	addr,cnt	Read Results of Region Detection
WRITE	0x03	addr,data	Equivalent to CAP_REGION for Compatibility
READ_REG	0x0C	addr,cnt	Read Camera Chip Registers
WRITE_REG	0x0D	addr,data	Write Camera Chip Registers (1)
CAP_REGION	0x0E	0,0	Capture and Find Color Regions (1)
RAW_SAMPLE	0x0F	0,0	Sample the Raw Image (used by GUI) (2)
LUT_MANAGE	0x10	0,0	Enter LUT Manage Mode (used by GUI) (2)
CAP_GRID	0x15	0,0	Capture and Compress using Gridding algorithm (1)
READ_GRID	0x16	addr,data	Read Results of the Gridding Algorithm (3)

(1) No return packets are generated for these instructions.
(2) Response is different from standard packets.
(3) The given address is internally multiplied by 16

Table 1. Available Instruction in HaViMo2.5

PING This instruction is used to check whether the device exists and is ready to receive the next instruction. The instruction returns an empty status packet.

READ_REGION This instruction is used to read the results of the region growing algorithm. This command accepts multi byte read. The data structure is described further in the data sheet.

WRITE This instruction is to achieve compatibility to Roboplus as it only supports *READ* and *WRITE* instructions. Therefore a write to an arbitrary address simulates a *CAP_REGION* instruction.

READ_REG This instruction is to read the content of camera registers. It is the same as *READ* instruction. This command accepts multi byte read.

WRITE_REG This instruction is to write the content of camera registers. It is the same as the *WRITE* instruction but it accepts only single byte write.

CAP_REGION This instruction starts capturing and processing of the next available frame. The processing algorithm used in this instruction is *Region Growing.* It takes approximately 50 ms to process a full frame. The main CPU should pole the functionality of the device using the *PING* command before sending the next instruction. The results can be then accessed using *READ_REGION* instruction.

RAW_SAMPLE With this instruction the camera module transmits a full frame of raw image data. This instruction is used when the GUI receives a request to sample a raw image. This instruction does not use the Standard packet protocol.

LUT_MANAGE After receiving this instruction, the module enters the programming mode, in this mode the device accepts no more packets, but other instructions assigned to manage the look-up table, such as erasing, reading and writing into it. This instruction is used by User interface during calibration phase and should not be used in implementation phase.

CAP_GRID This instruction invokes the gridding algorithm. The algorithm compresses the image into a 32*24 cell grid and reports the number of pixels and the color observed in the 5x5 window related to the cell. Only one color is accepted for each cell. Lower color codes dominate the higher ones but Unknown = 0 has the lowest priority.

READ_GRID This instruction reads the results of the gridding algorithm. It has a similar structure to other READ instructions but the given address is internally multiplied by 16. This gives access to a wider range of addresses, however at least 16 bytes should be read to cover the whole space. The data structure is described further in the data sheet.

3.3 Image Processing Algorithms

HaViMo2.5 is equipped with two image processing algorithms, which are described in this section. In the first step both algorithms translate color values to object codes using the built-in look-up table. Therefore an exact calibration of the colors should have a great impact on the results of the recognition.

3.3.1 On-line Region Growing Algorithm

The goal of the region growing algorithm is to detect contiguous color blobs in the image. A 4-pixel neighborhood is used to determine connections. The function is invoked using the instruction *CAP_REGION*. The detected regions are summarized using the following parameters:

Value	Bytes	Description
Index	1	Contains zero if the region is invalid and nonzero otherwise.
Color	1	Color code of the detected region (0 = Unknown, 1 = Ball , ...)
Pixels	2	Number of detected pixels inside the region
SumX	4	Sum of the X coordinates of the detected pixels (*)
SumY	4	Sum of the Y coordinates of the detected pixels (*)
MaxX	1	Bounding box right margin
MinX	1	Bounding box left margin
MaxY	1	Bounding box bottom margin
MinY	1	Bounding box top margin

(*) Values can be divided by the number pixels to obtain the center point.

Table 2. Data Format of the Results of Region Growing Algorithm

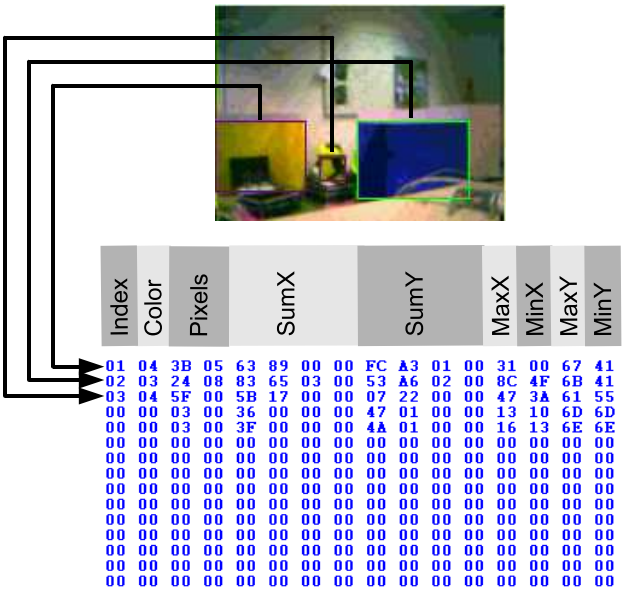
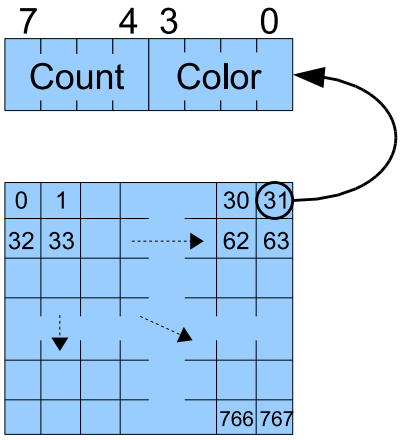


Figure 5. Results of the Region Growing Algorithm

To access the results, the instruction *READ_REG* should be used. Up to 15 regions can be read from the address range 0x10 to 0xFF using the *READ_REG* (0x02) instruction. Following example shows the produced output of the module according to a given image.

3.3.2 On-line Gridding Algorithm

After invoking the gridding algorithm with *CAP_GRID* instruction, it compresses the image into a grid of 32*24. Each cell is a representative for a 5x5 square on the original image. For each cell one byte is calculated which contains the flowing information. The lower 4 bits determine which color was the most prior detected in the square. The higher 4 bits are the number of pixels occupied with this color. If over 15 pixels are detected the count remains 15.



The algorithms results 768 bytes of data. These can be read using the *READ_GRID* instruction. Note that the given address is internally multiplied by 16 to increase the access range. It is therefore required that a multi-byte read operation with at least 16 bytes be used.

Following example shows the result of a the gridding algorithm. Note that only the color is visualized.

3.4 Calibration Fucntions

In order to calibrate the device, there are several instructions reserved for making a raw sample form the camera image, and accessing the look-up table. These section describes how these instructions are used.

3.4.1 Downloading a Raw Image Sample

The instruction RAW_SAMPLE (0x0F) is used to get a complete frame of camera image form the module. As the result, 19200 bytes of data are sent back over the bus. The amount of data corresponds a full frame of 160*120 pixels in YCrCb 422 format, however it is dereived form several consequtive frames. This is to reduce the data rate and avoid a buffer over flow in systems with low speed serial communications. In *RAW_SAMPLE* instruction the image sample is obtained from 40 consequtive frames. The correspondence is described in Figure 6.

0,0	1,0	2,0	3,0	
0,1	1,1	2,1	3,1	
0,2	1,2	2,2	3,2	1st Frame
...	...	...	...	
0,119	1,119	2,119	3,119	
4,0	5,0	6,0	7,0	2nd
4,1	5,1	6,1	7,1	Frame
...	...	...	...	...
156,0	157,0	158,0	159,0	
...	...	...	...	40th Frame
156,119	157,119	158,119	159,119	

Figure 6. Downloaded data stream and its correspondence to the image using *RAW_SAMPLE*.

bits	7-4	3-0
Even Addresses	Y	Cr
Odd Adresses	Y	Cb

Figure 7. YCrCb 422 data format

3.4.2 Color Look-up Table Access

By invoking a LUT_MANAGE command, the module enters its look-up table access mode. The request is replied with a '*' character. In this mode normal commands are disabled and a new set of commands are activated to access a part of the flash, where the color look-up table resides.

Instruction	Val	Par. #	Function	Returns
ERASE	'e'	0	Erase the color look-up table	0x0D
SET_ADDR	'a'	2	Set the current address pointer	0x0D
WRITE_WORD	'w'	2	Write 16 bits of data in the page buffer	0x0D
WRITE_PAGE	'm'	0	Write page buffer into flash	0x0D
READ_WORD	'r'	0	Read 16 bits of data from the color look-up table	B1 B2 0x0D
EXIT_LUTMAN	'x'	0	Exit from LUT manage mode	Status Packet

Table 1. Instruction in LUT_MANAGE mode

Note that due to the access to the flash, some commands may need several of milliseconds to complete. It is therefore necessary that the client waits for an acknowledge form the module.

Valid address range is between 0x0100 and 0x10FF. Values outside this range are capped internally.