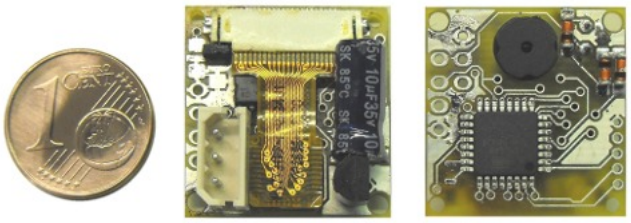
HaViMo2

Image Processing Module

March 13, 2010



Features

- Integrated Color CMOS Camera
  - Frame Resolution: 160*120 Pixels
  - Color Depth: 12 bits YCrCb
  - Frame Rate: 19 Fps
  - Full Access to all CMOS Camera registers
    - Saving values in EEPROM, 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 Griding
        * 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
        * Interlaced Output at 19 FPS
        * Full Frame Output at 0.5 FPS

- Supported Hardware
    - Half Duplex (*ROBOTIS*)
        * CM5
        * CM510
        * USB2Dynamixel
    - Full Duplex (*RoboBuilder*)
        * RBC
    - Requirements for other platforms
        * TTL level RS232
        * 115200 BAUD for Full Duplex Mode (*RoboBuilder* Mode)
        * 1 MBAUD for Half Duplex Mode (*ROBOTIS* Mode)

- Supported Software
    - Roboplus (*ROBOTIS*)
    - Direct C programming (*ROBOTIS*)
    - RBC firmware
    - Direct C programming (*RoboBuilder*)

1 Introduction

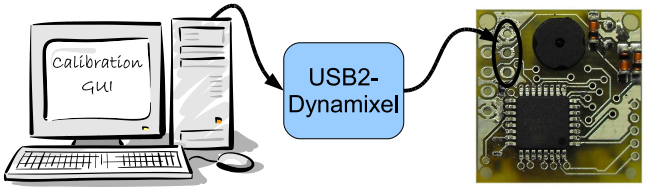
HaViMo is a computer vision solution for low power microprocessors. It is equipped with a CMOS camera chip and a microcontroller which performs the image processing. The results are then accessed via serial port. In HaViMo2 several features such as frame rate are improved.

In addition to the region growing algorithm available in prior versions, a new algorithm is implemented called *Griding*. The algorithm is a suitable pre-processing step for many other applications such as object recognition and self localization.

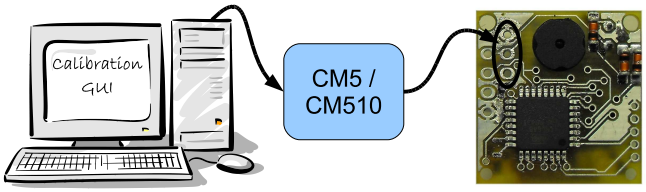
The compatibility area of the module is also extended to more software and hardware platforms. HaViMo2 is compatible with *ROBOTIS* and *RoboBuilder* bridges. It can also be integrated in programs developed in Roboplus. Other platforms can also communicate with the module using either half or full-duplex serial protocols.

2 Hardware Setup

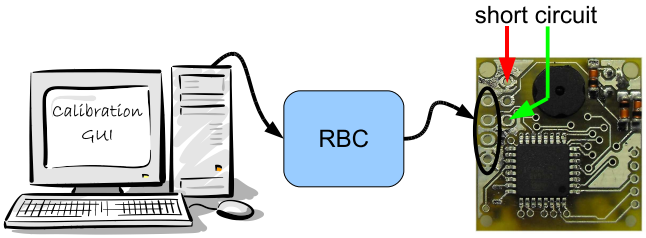
The module can be used in different configurations according to the hardware platform it is used in conjunction with. These are described in calibration and implementation modes.



a) Direct Calibration using USB2Dynamixel



b) In-System Calibration in *ROBOTIS* Mode



c) In-System Calibration in *RoboBuilder* Mode

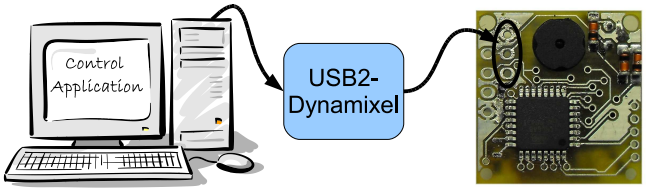
Figure 1. Possible Hardware Configurations in Calibration Mode.

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.

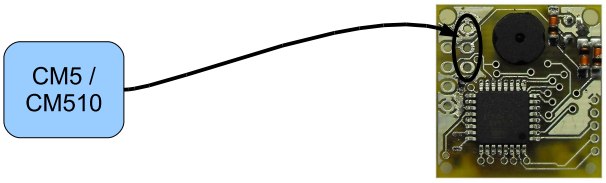
Possible configurations in calibration modes are:

- Direct Calibration using USB2Dynamixel
- The module is connected through USB2Dynamixel to a PC. Note that it is necessary to connect the power supply externally to the device.
- In-System Calibration in *ROBOTIS* Mode
  - The module is connected through a standard *ROBOTIS* bridge (CM5, CM510) to a PC. The firmware of the bridge allows data communication between HaViMo2 and the PC

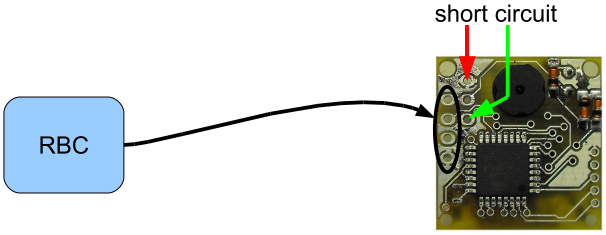
- In-System Calibration in *RoboBuilder* Mode
  - The module is connected through a standard *RoboBuilder* bridge (RBC) to a PC. The firmware of the bridge allows data communication between HaViMo2 and the PC. To enter this mode the marked pins on the PCB should be jumpered prior to module start-up.



a) Implementation on a PC Platform using USB2Dynamixel



b) Implementation in *ROBOTIS* Mode



c) Implementation in *RoboBuilder* Mode

Figure 2. Possible Hardware Configurations in Implementation Mode

3 Function Description

HaViMo2 is accessed on a serial bus. A communication protocol is designed for accessing the device which 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 supports both half and full duplex communications in physical layer. This makes the module compatible with both *ROBOTIS* and *RoboBuilder* platforms. However to avoid conflicts

with other members of the bus, it is also necessary to support the communication protocols of both platforms. Following diagram shows a summary of command and response packets in both operation modes.

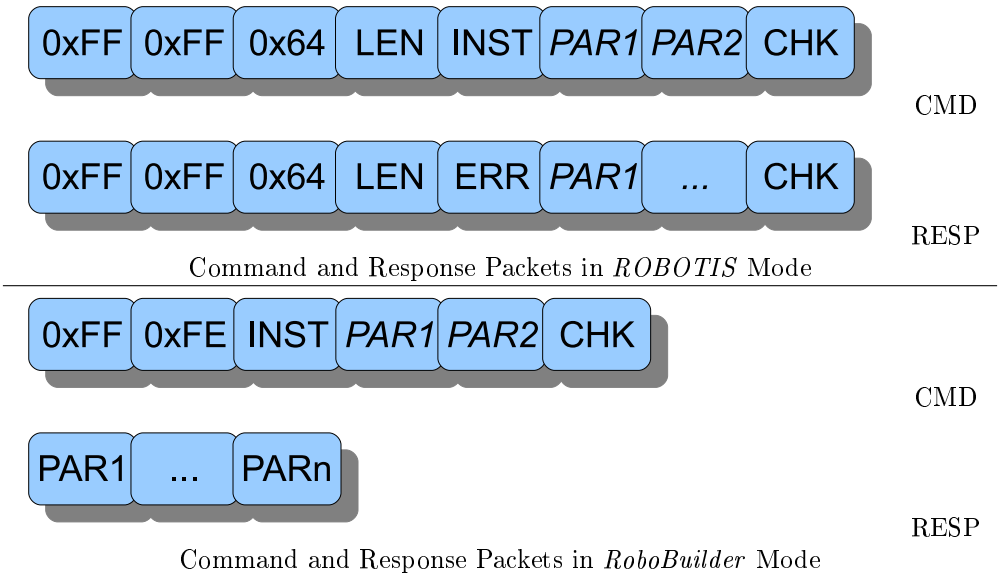


Figure 3. Command and Response Packets in *ROBOTIS* and *RoboBuilder* Modes

3.1.1 Communication Protocol in *ROBOTIS* Mode

To understand the communication protocol in *ROBOTIS* mode, it is recommended to read *Dynamixel AX-12* data sheet. The Command packet is structured as follows.

Header 2 times 0xFF.

ID Fixed on 0x64 = 100.

LEN Number of bytes to be further transmitted.

INST Instruction code described in table 1.

PARn Optional parameters passed to the instruction.

CHK Check sum is calculated as complement of the lowest 8 bits of the sum of all bytes in the packet excluding the header.

The response packet has a similar structure as the command packet, however the instruction is replaced with an error indicator and parameters are filled with the results of running the instruction.

3.1.2 Communication Protocol in *RoboBuilder* Mode

To understand the communication protocol in *RoboBuilder* mode, it is recommended to refer to the *RoboBuilder wCK* data sheet. The idea behind *RoboBuilder* compatibility mode is to simulate

a *RoboBuilder* configuration packet being transmitted to ID = 30 which is not existing on the bus, so that it is ignored by the other bus members.

The command packet is structured as follows.

Header The sequence 0xFF 0xFE, which also includes the fixed ID = 30

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 instruction in HaViMo2.

Instruction	Value	No. of Param.	Function
PING	0x01	0	No action. Used for obtaining a Status Packet
READ_REGION	0x02	2	Read Results of Region Detection
WRITE	0x03	2	Equivalent to CAP_REGION for Compatibility
READ_REG	0x0C	2	Read Camera Chip Registers
WRITE_REG	0x0D	2	Write Camera Chip Registers (1)
CAP_REGION	0x0E	0	Capture and Find Color Regions (1)
RAW_SAMPLE	0x0F	0	Sample the Raw Image (used by GUI) (2)
LUT_MANAGE	0x10	0	Enter LUT Manage Mode (used by GUI) (2)
RD_FILTHR	0x11	2	Read Noise Filter Thresholds
WR_FILTHR	0x12	2	Write Noise Filter Thresholds (1)
RD_REGTHR	0x13	2	Read Region Filter Thresholds
WR_REGTHR	0x14	2	Write Region Filter Thresholds (1)
CAP_GRID	0x15	0	Capture and Compress using Griding algorithm (1)
READ_GRID	0x16	2	Read Results of the Griding Algorithm (3)
SAMPLE_FAST	0x17	0	Fast Sample (used by GUI) (2) (4)

(1) No return packets are generated for these instructions.
(2) Response is different from *ROBOTIS/RoboBuilder* standard packets.
(3) The given address is internally multiplied by 16
(4) Not supported in *RoboBuilder* Mode

Table 1. Available Instruction in HaViMo2

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 60 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.

RD_FILTHR, WR_FILTHR These instructions make access to the threshold values of the noise filter. Noise filter thresholds are actually the minimum number of neighbor pixels in a scan line which should be counted as a part of a region. For each color a separate 8-bit threshold should be defined. Default values are 8 for unknown color and 2 for others. The address field contains the index of the color category (0 = unknown,...). The structure of these instructions are the same as in *READ* and *WRITE*, however multi-byte read/write is not supported.

RD_REGTHR WR_REGTHR These instructions provide access to the threshold values of the region filter. The region filter thresholds define the minimum number of pixels inside a region, regions with fewer pixels are filtered away. For each color a separate 16-bit threshold should be defined. The address field contains 2*index of the color category (0 = unknown,...). Default values are (0,5,50,50,50,100,50,50). No region is built for color 0 (unknown). The structure of these instructions are the same as in *READ* and *WRITE*, however multi-byte read/write is not supported.

CAP_GRID This instruction invokes the griding 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 griding 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.

SAMPLE_FAST This instruction is used to download a RAW image from the module. Unlike *RAW_SAMPLE* instruction, the image is transferred with full baud rate, 1Mbps. Therefore it is not possible to use the instruction with low baud rates such as in *RoboBuilder* mode.

3.3 Image Processing Algorithms

HaViMo2 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.

Figure 3. Data Format of the Results of Region Growing Algorithm

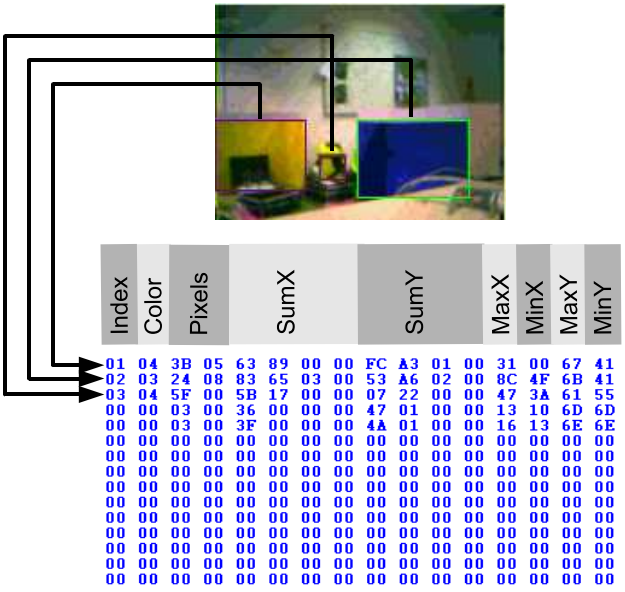
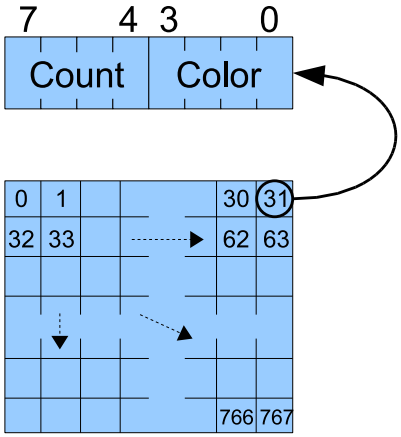


Figure 4. Results of the Region Growing Algorithm

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 Griding Algorithm

After invoking the griding 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 griding algorithm. Note that only the color is visualized.