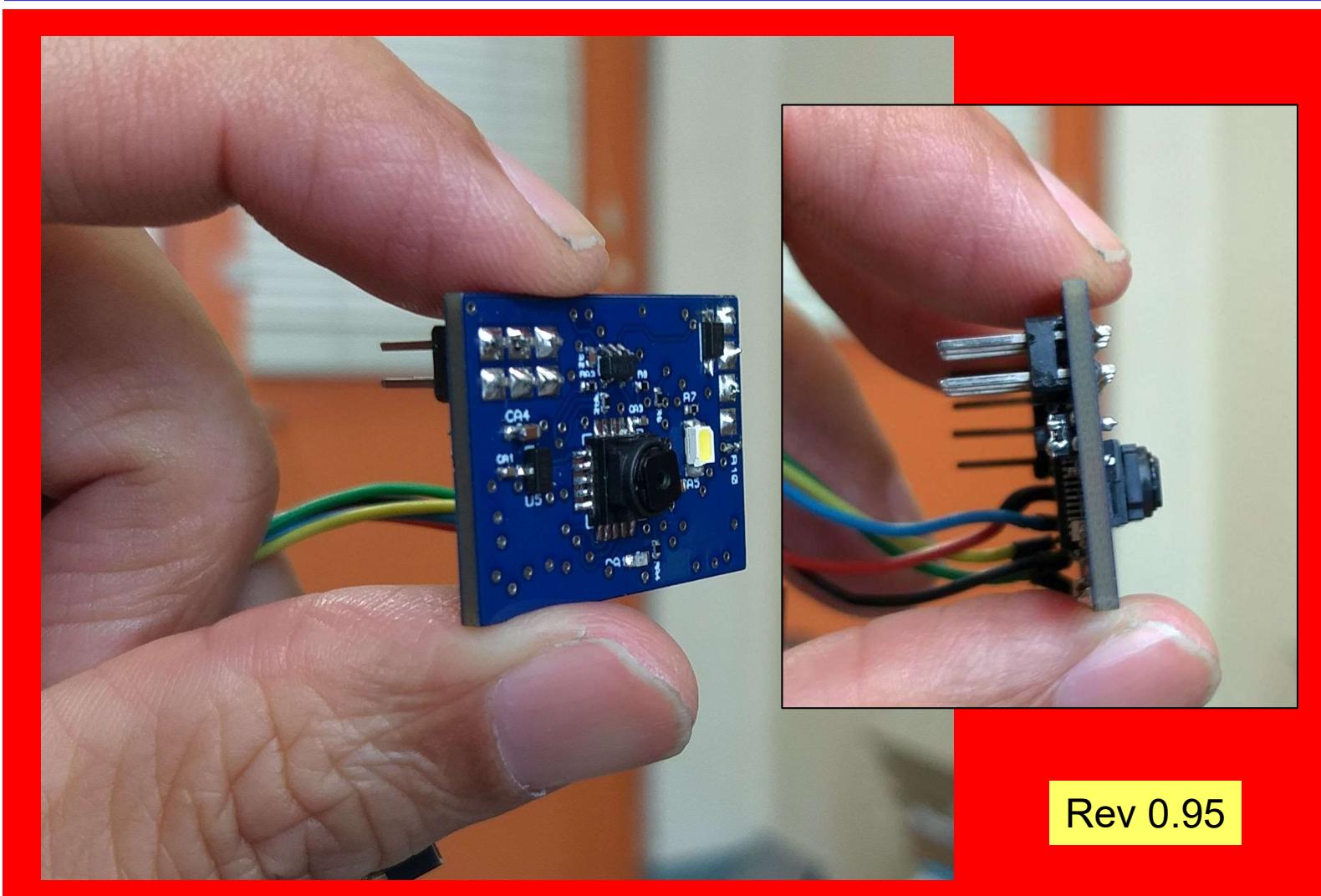


# MVM V1.5C Quick Start Guide



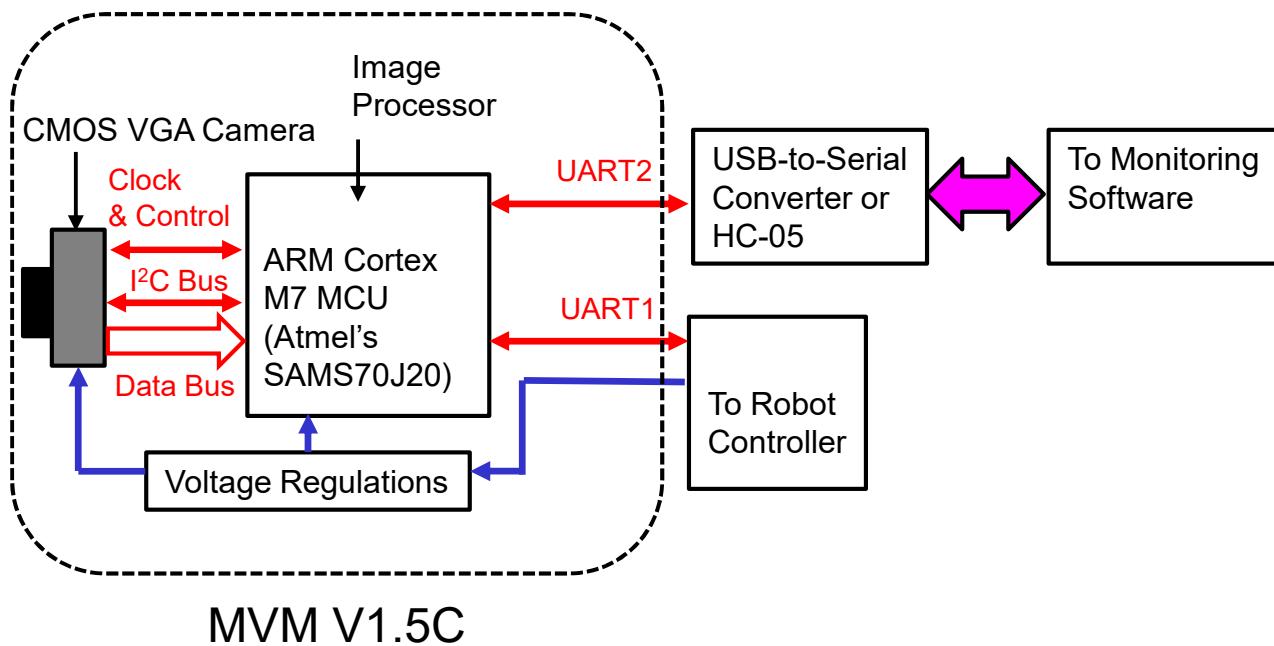
Rev 0.95

# What Is It?

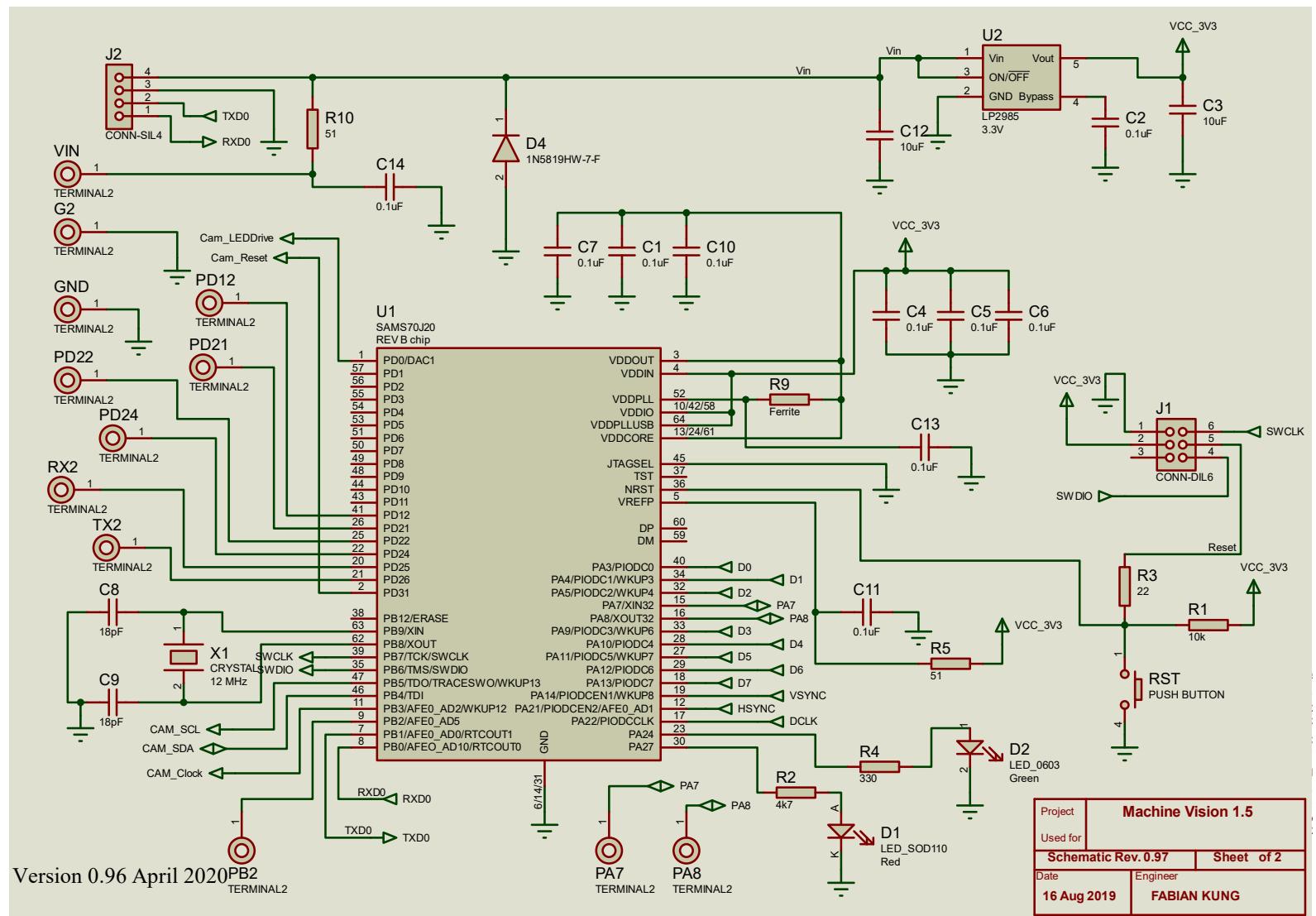
---

- An open source easy to use low resolution CMOS camera with on-board real-time image processing.
- Requires 5V, 150 mA power source, and interface through UART port.
- Support 160x120 pixels (QQVGA) and 320x240 pixels (QVGA) color image at 20 frames-per-second.
- Current image processing algorithm:
  - Edge detection via Sobel kernel.
  - Bright spot detection.
  - Obstacle detection using luminance contrast.
  - Color detection.
- Coming soon:
  - Line following.
  - Optical flow.
  - Neural-network (no guarantee!)

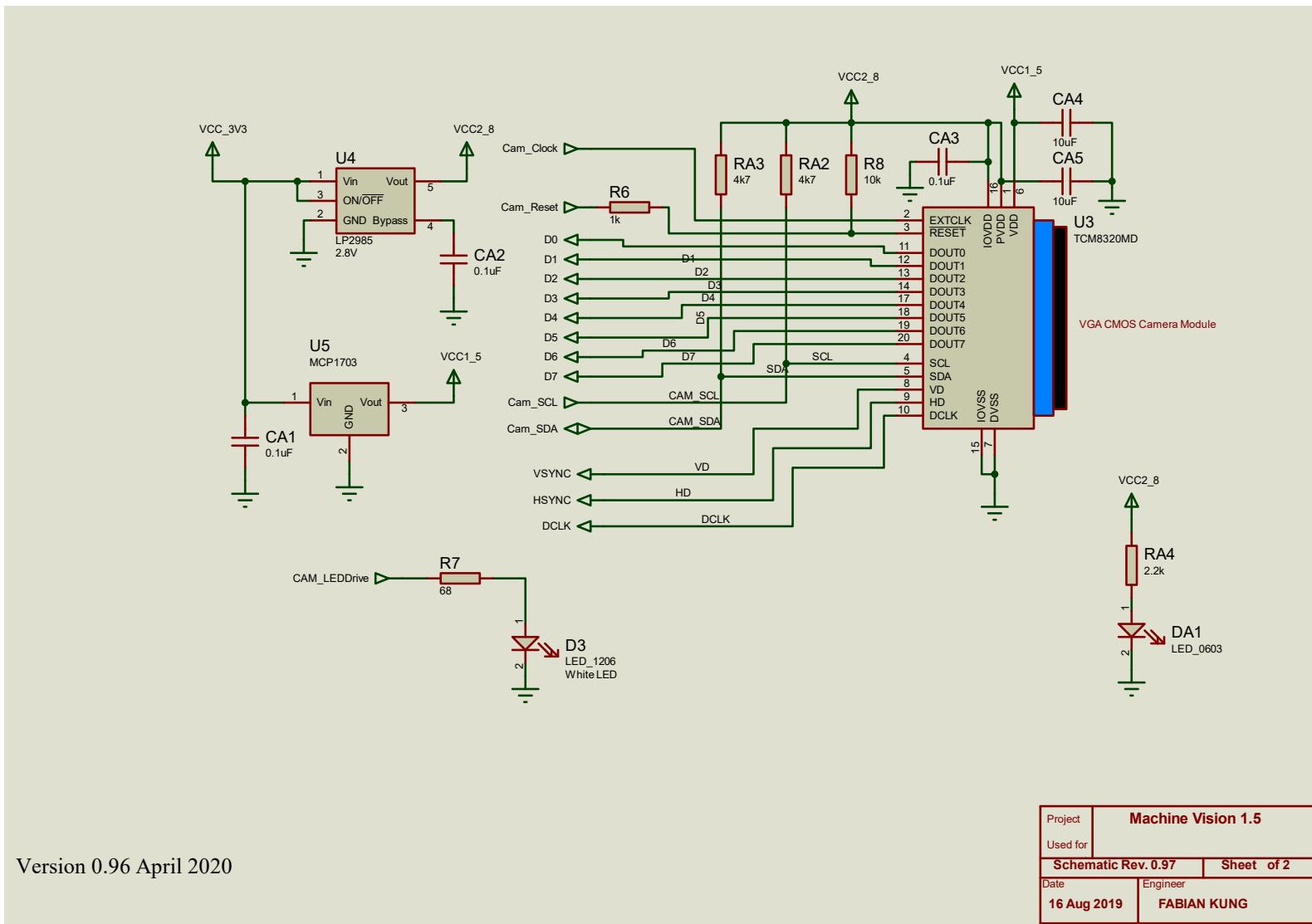
# Block Diagram



# Schematic 1 – Micro-controller Core



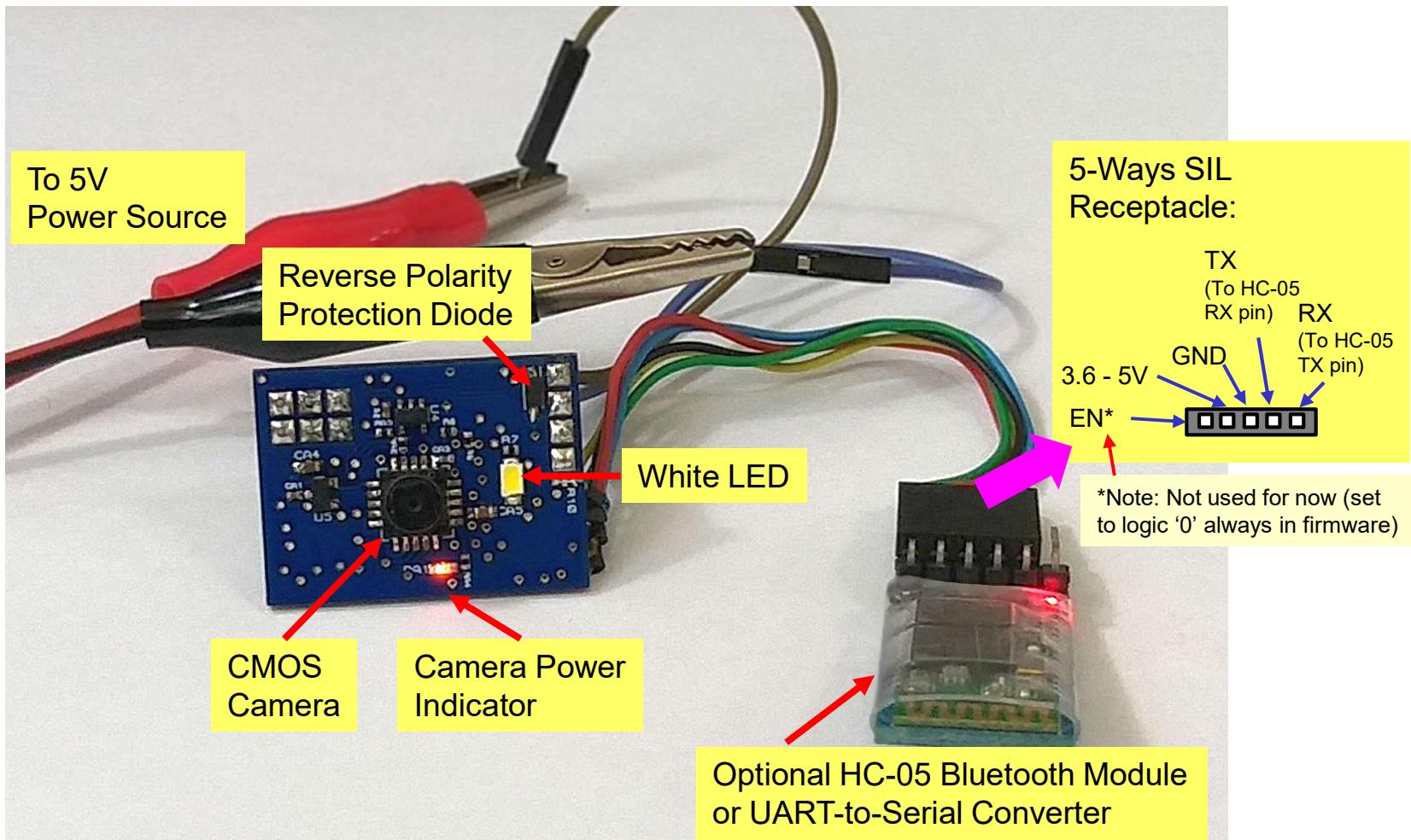
# Schematic 2 – Camera Sub-Circuit



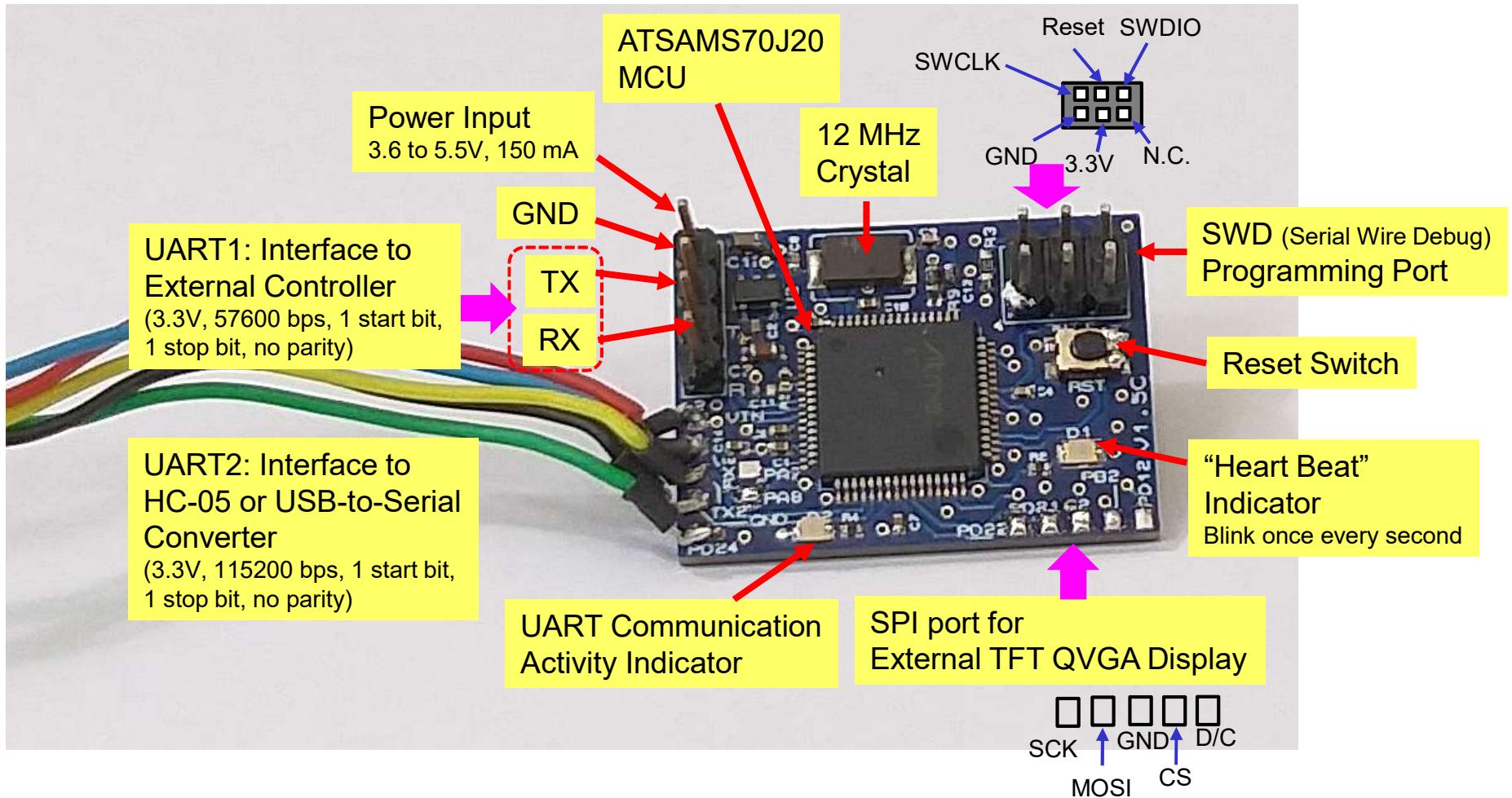
Version 0.96 April 2020

Project	Machine Vision 1.5	
Used for		
Schematic Rev. 0.97	Sheet of 2	
Date	Engineer	
16 Aug 2019	FABIAN KUNG	

# Rear View (MVM V1.5C)



# Front View (MVM V1.5C)



# Files

---

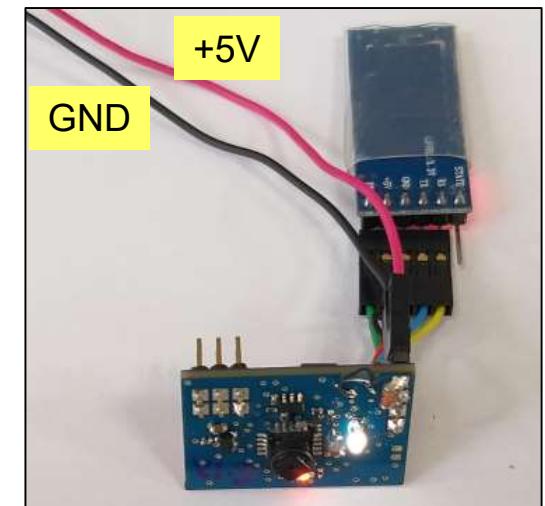
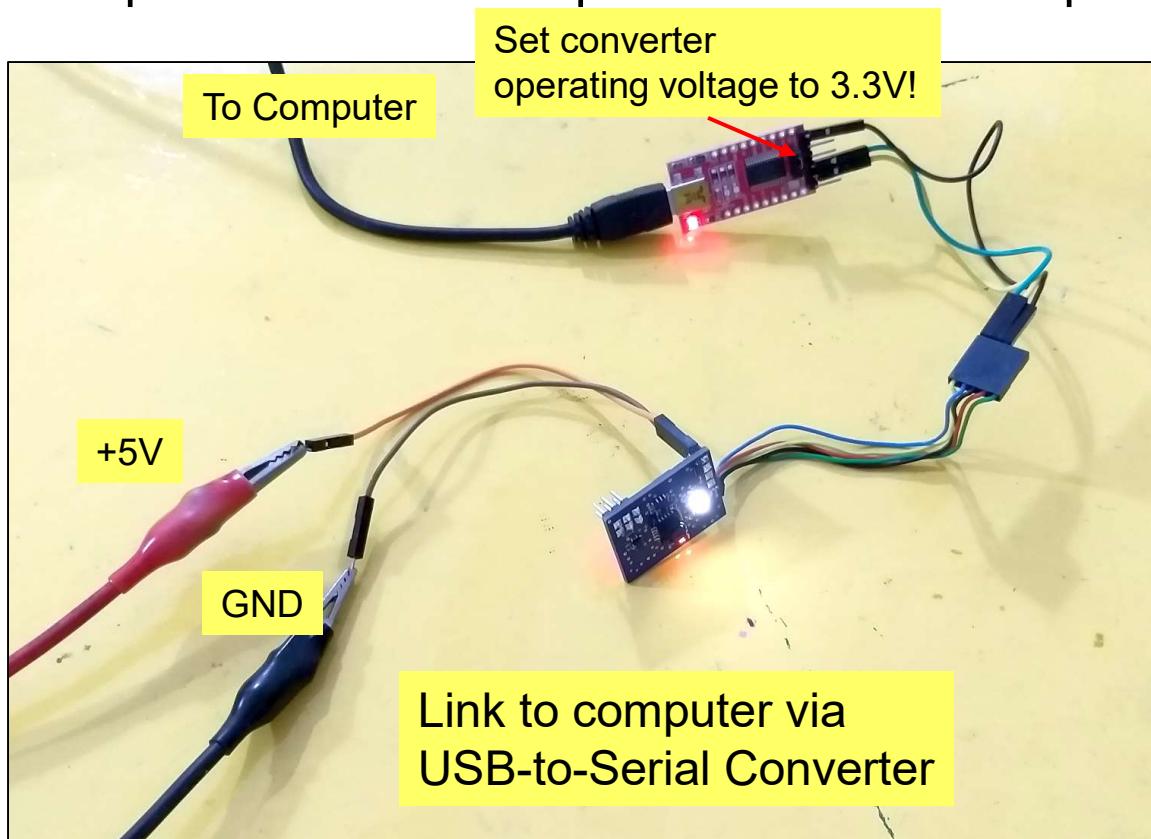
- All relevant files can be obtained from  
[https://github.com/fabiankung/MVM\\_V1\\_5C](https://github.com/fabiankung/MVM_V1_5C)
- Firmware is build using **Atmel Studio 7**.
- PC software is build using **Visual Studio Community 2017** or later.

---

# **Observing the Camera Image via Machine Vision Monitor Software**

# Step 1 – Power Up the MVM

- Here we assume the MVM is connected to HC-05 Bluetooth wireless module or a USB-to-Serial Converter, as shown in the various implementation examples below. Power up the module.



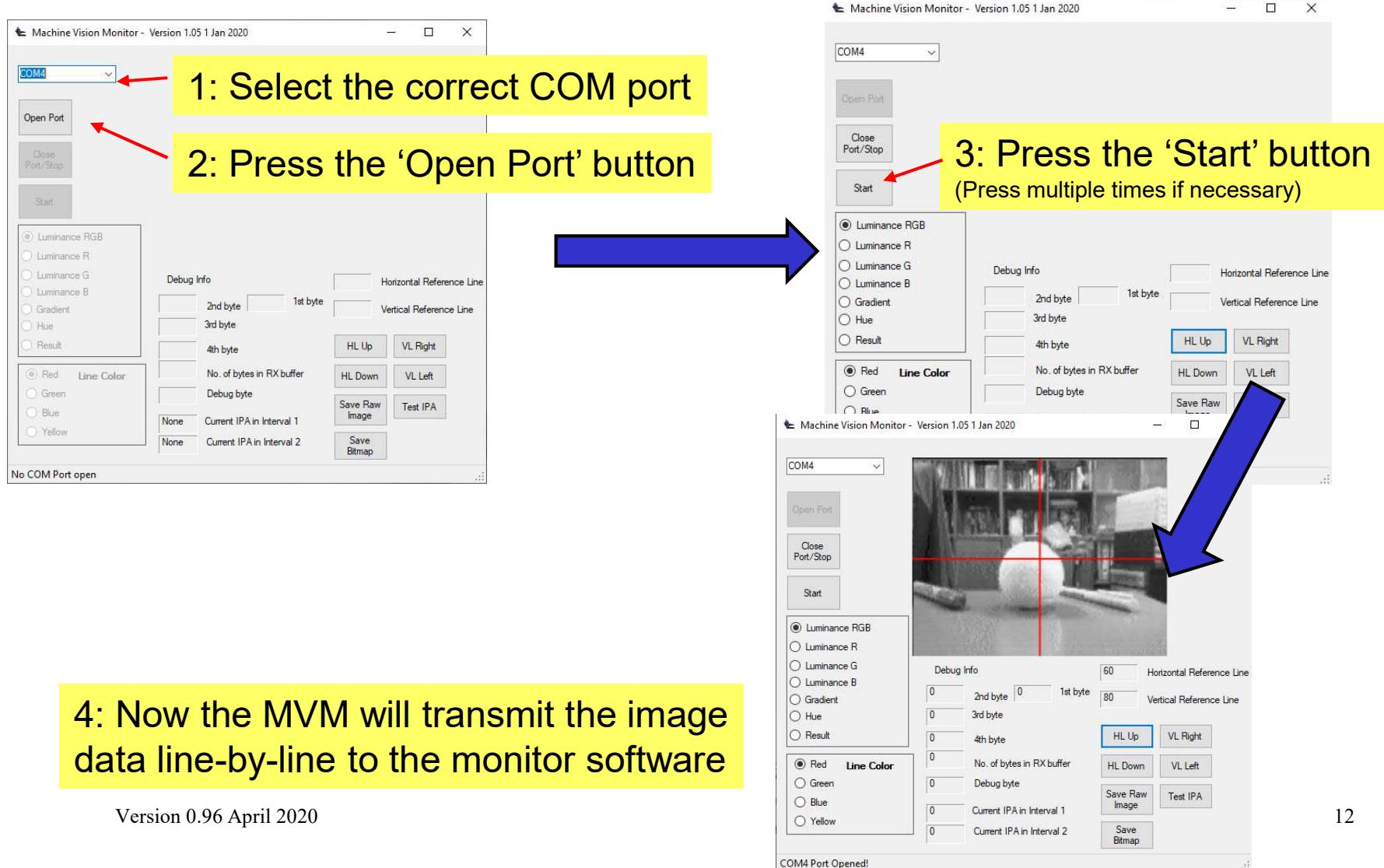
Link to computer via  
HC-05 Bluetooth Module

## Step 2 – Pair Computer to HC-05

---

- If need to pair the computer to HC-05.
- Then check virtual COM port number on the computer (for instance by going to the Device Manager in Windows).

# Step 3 – Run the Machine Vision Monitor Software (MV\_Monitor.exe)

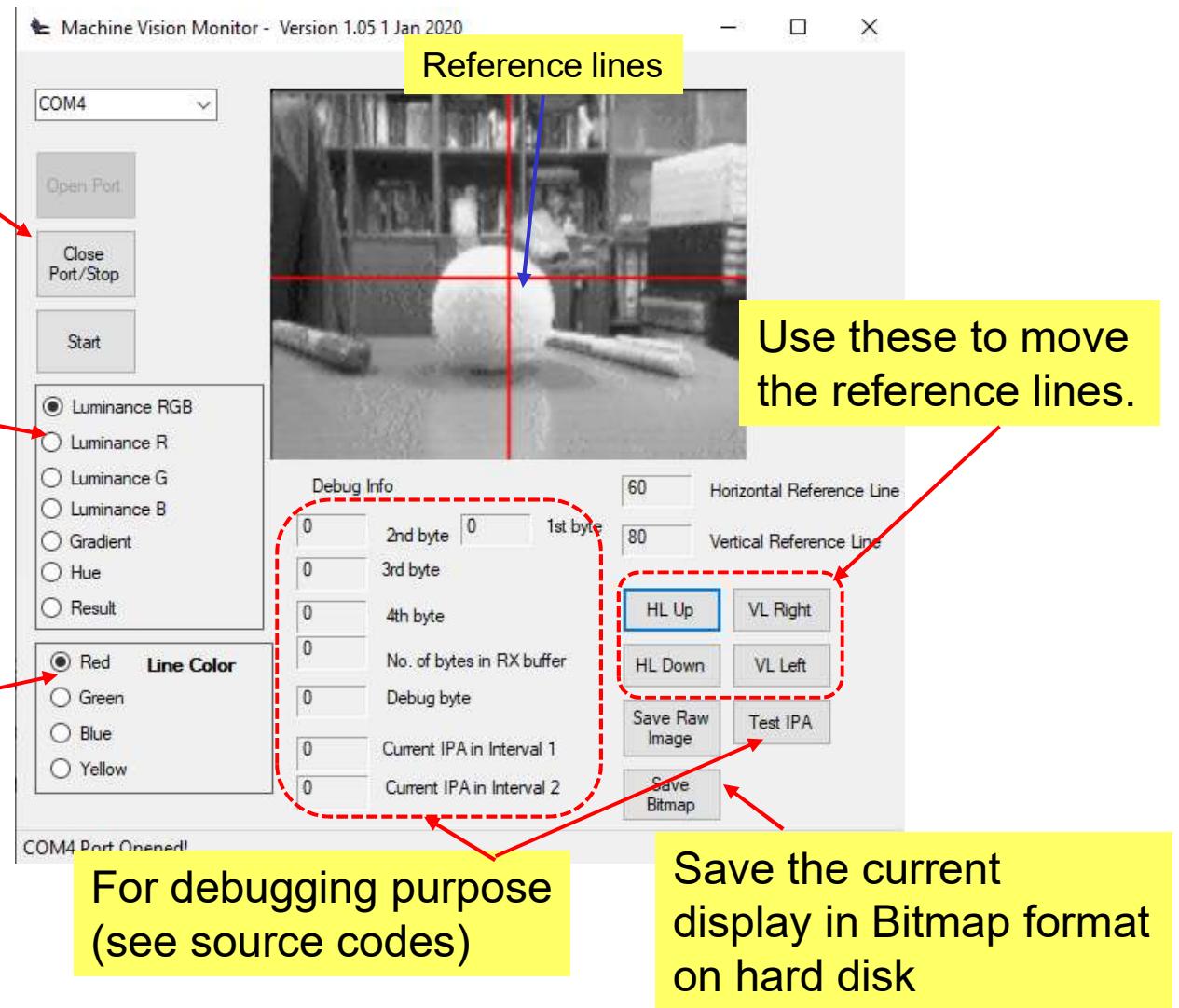


# Other Information (1 of 2)

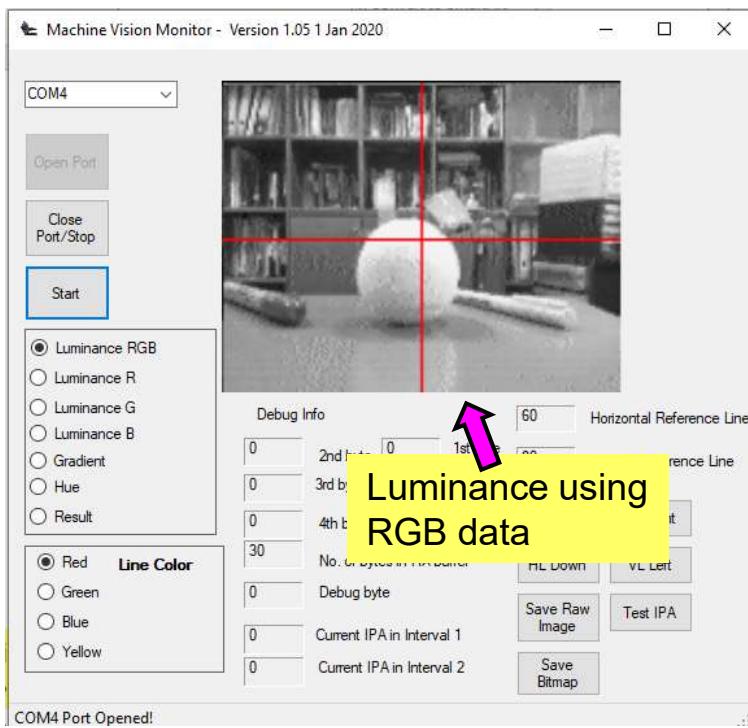
Always close the COM port before shutting down the software

Select between various display options. All these are processed in the MVM. (See next slide)

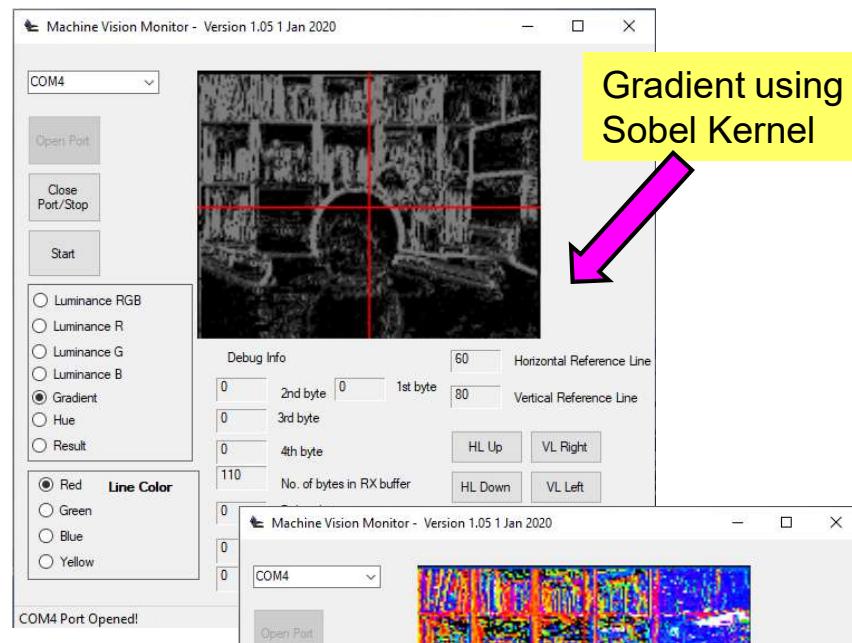
Change reference line color.



# Other Information (2 of 2)

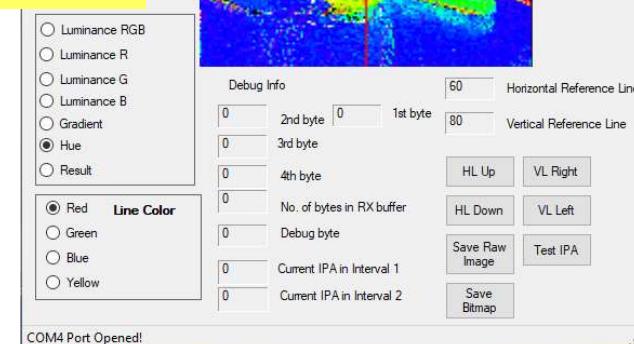


Luminance using  
RGB data



Gradient using  
Sobel Kernel

Hue extracted  
From RGB data

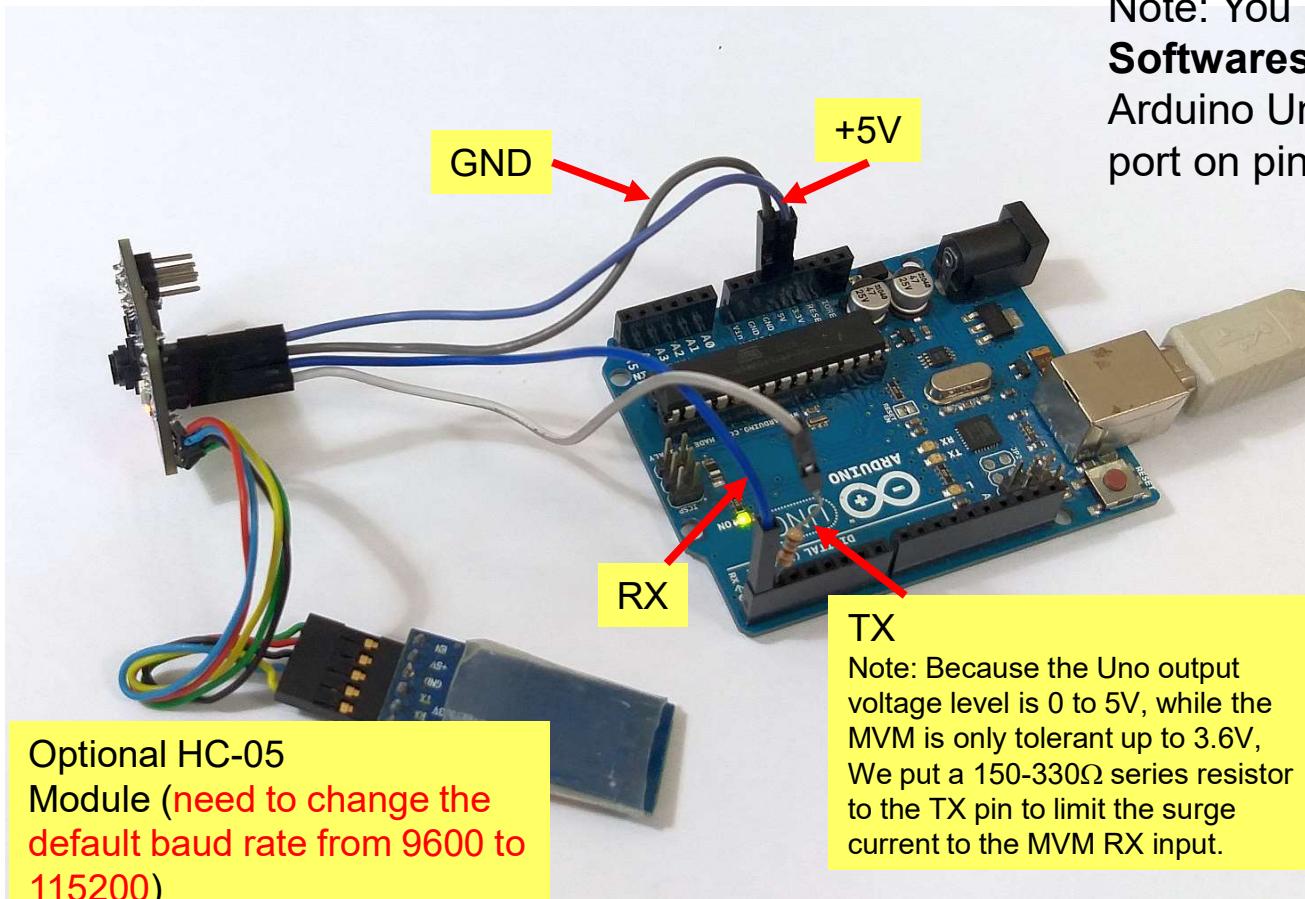


---

# **Connection to External Controller for Robotic Projects**

# Connection to External Controllers

- Here we use an Arduino Uno to demonstrate the connection.



# UART1 Communication Protocol

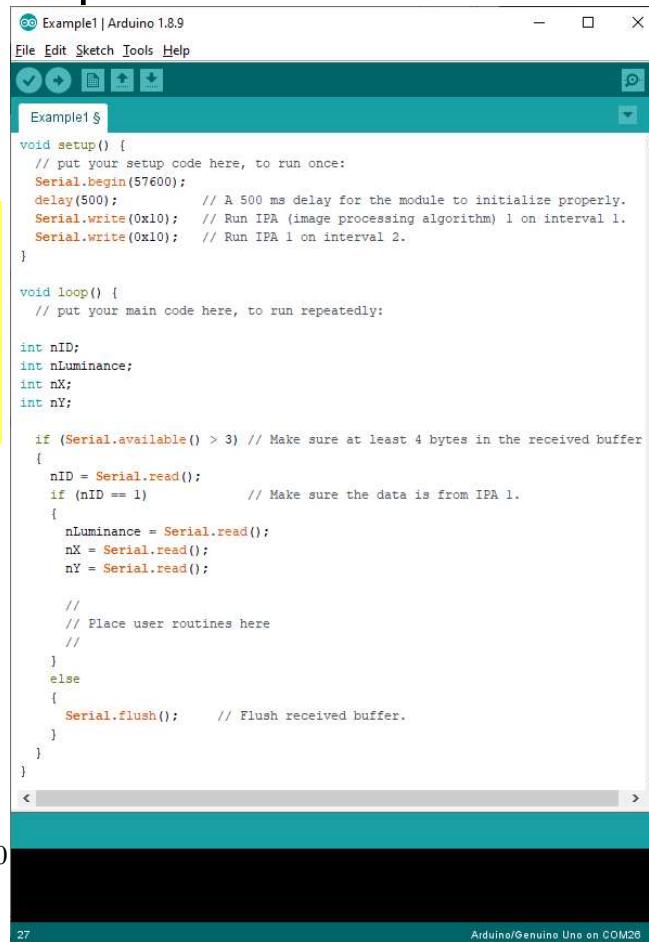
---

Image Processing Algorithm (IPA)	To Activate	MVM Output
Search for brightest spot in a scene. Image resolution = 160x120	Send hex values to MVM: 0x10 to search for brightest spot	4 bytes: Byte 1 = 1 (Algorithm ID) Byte 2 = Maximum luminance value (1 to 127). Byte 3 = x coordinate of region Byte 4 = y coordinate of region
Obstacle detection on lower half of the image. Image resolution = 160x120	Send hex value to MVM: 0x20	4 bytes: Byte 1 = 2 Byte 2 = 0b00000b <sub>2</sub> b <sub>1</sub> b <sub>0</sub> Byte 3 = 0b00000b <sub>2</sub> b <sub>1</sub> b <sub>0</sub> Byte 4 = 0b00000b <sub>2</sub> b <sub>1</sub> b <sub>0</sub>
Color object detection. Image resolution = 160x120	Send hex values to MVM: 0x30 for yellow-green object 0x31 for red object 0x32 for green object 0x33 for blue object	4 bytes: Byte 1 = 3 Byte 2 = Number of pixels matched Byte 3 = x coordinate of region Byte 4 = y coordinate of region

# Example 1 – Activate Search for Brightest Spot Algorithm

- Assume the MVM is connected to an Arduino Uno. The left panel shows a simple Arduino Sketch to activate the image processing algorithm to search for brightest spot on both **Interval 1** and **2**, giving effective response time of 50 ms.

Note:  
See Appendix for  
Another version  
of this code using  
SoftwareSerial



```
Example1 | Arduino 1.8.9
File Edit Sketch Tools Help
Example1.ino
void setup() {
    // put your setup code here, to run once:
    Serial.begin(57600);
    delay(500);           // A 500 ms delay for the module to initialize properly.
    Serial.write(0x10);   // Run IPA (image processing algorithm) 1 on interval 1.
    Serial.write(0x10);   // Run IPA 1 on interval 2.
}

void loop() {
    // put your main code here, to run repeatedly:

    int nID;
    int nLuminance;
    int nX;
    int nY;

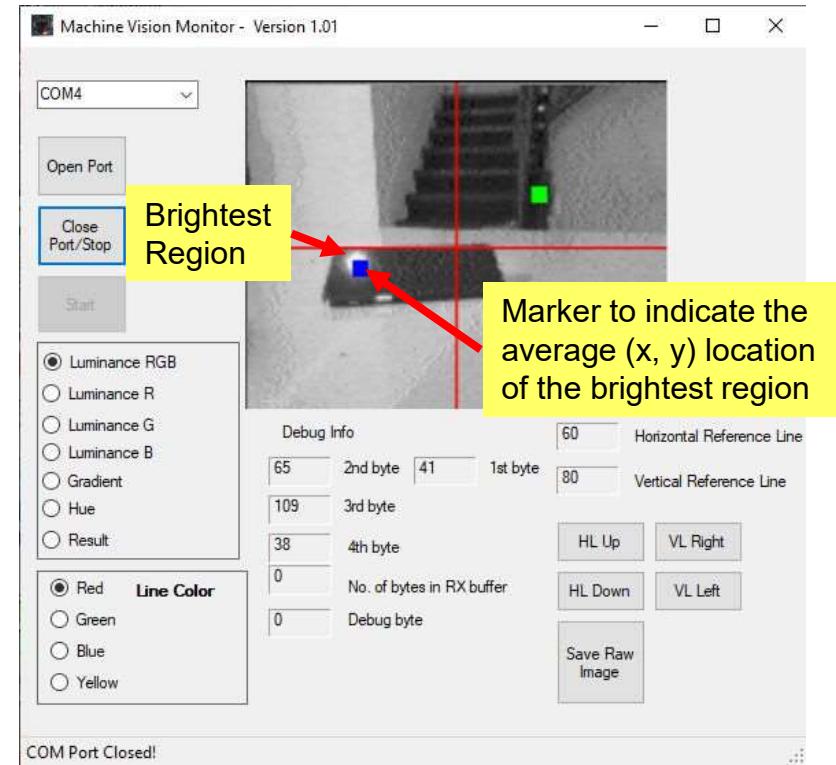
    if (Serial.available() > 3) // Make sure at least 4 bytes in the received buffer
    {
        nID = Serial.read();
        if (nID == 1)          // Make sure the data is from IPA 1.
        {
            nLuminance = Serial.read();
            nX = Serial.read();
            nY = Serial.read();

            // Place user routines here
            //
        }
        else
        {
            Serial.flush();    // Flush received buffer.
        }
    }
}
```

Version 0.96 April 2020

27

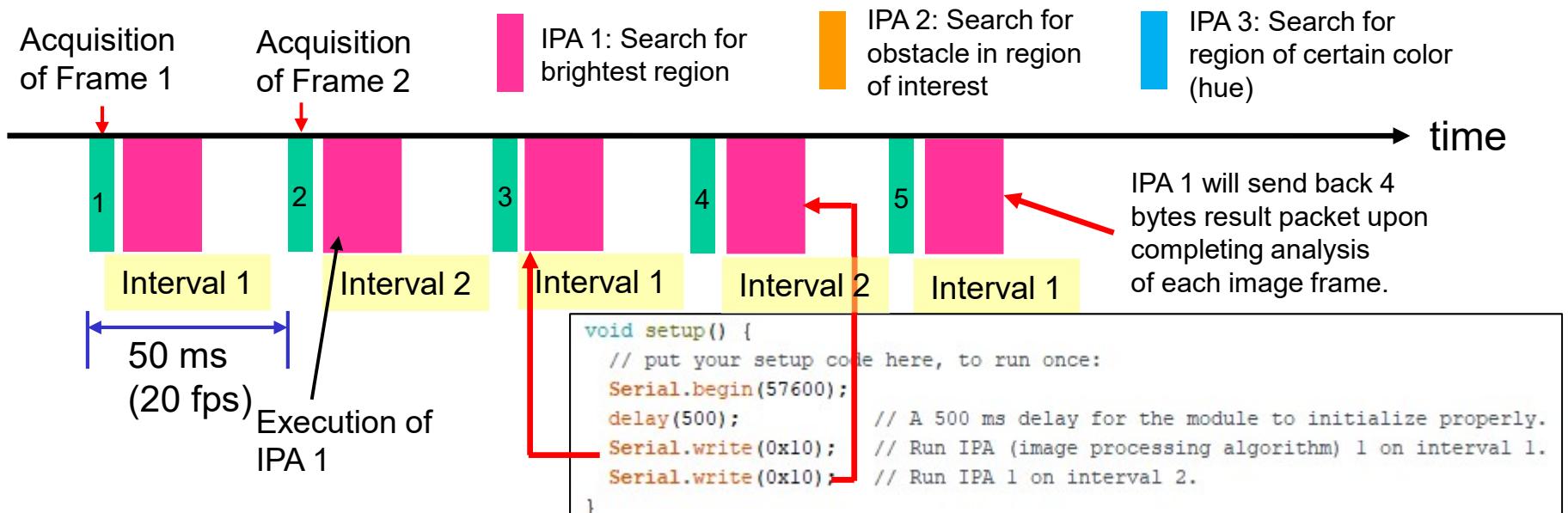
Arduino/Genuino Uno on COM26



18

# Example 1 - More on 'Interval'

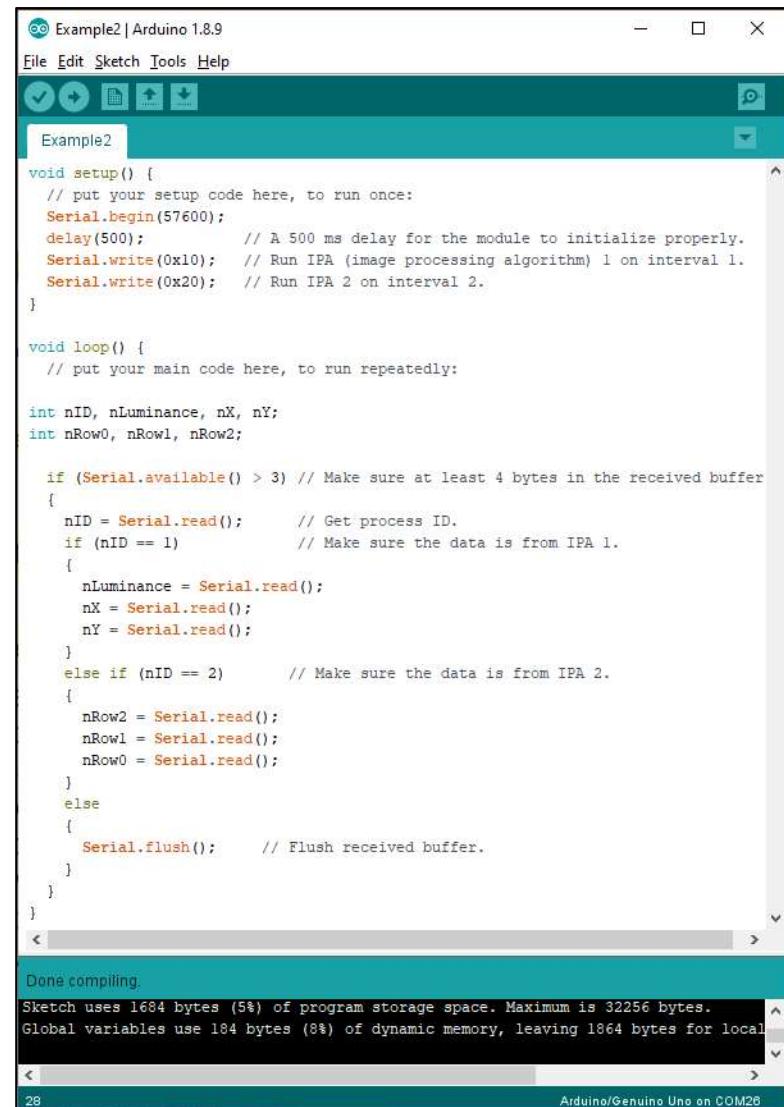
- The firmware of MVM V1.5C assigns odd image frames to *Interval 1* and even image frames to *Interval 2*.
- An image processing algorithm (IPA) can be attached to each interval as shown below and executed after acquisition of a new image frame.



- The C code snippet attaches IPA 1 to both Interval 1 and Interval 2 of the execution flow, thus in this setting IPA 1 runs every 50 ms and any changes in scene is detected within 50 ms.

# Example 2 - Activate Both Search for Brightest Region (IPA 1) and Obstacle (IPA 2) Algorithms

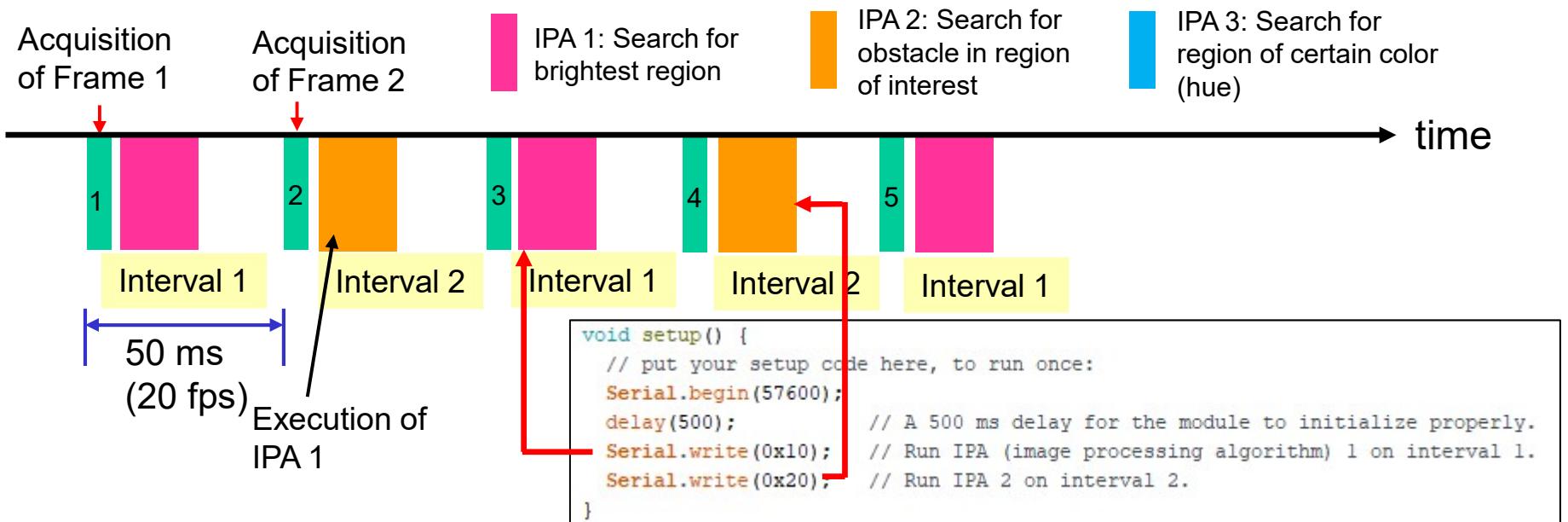
- In this example we attach IPA 1 to Interval 1 and IPA 2 to Interval 2.
- Thus a robot using the MVM V1.5C can be programmed to move towards a bright light source while at the same time avoid any obstacle on the floor.



The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** Example2 | Arduino 1.8.9
- Menu Bar:** File Edit Sketch Tools Help
- Sketch Area:** The code for the Example2 sketch is displayed. It includes setup() and loop() functions. The setup() function initializes the serial port and runs IPA 1 and IPA 2. The loop() function reads data from the serial buffer to determine if it's from IPA 1 or IPA 2 and processes it accordingly.
- Status Bar:** Done compiling.  
Sketch uses 1684 bytes (5%) of program storage space. Maximum is 32256 bytes.  
Global variables use 184 bytes (8%) of dynamic memory, leaving 1864 bytes for local variables.
- Bottom Bar:** Arduino/Genuine Uno on COM26

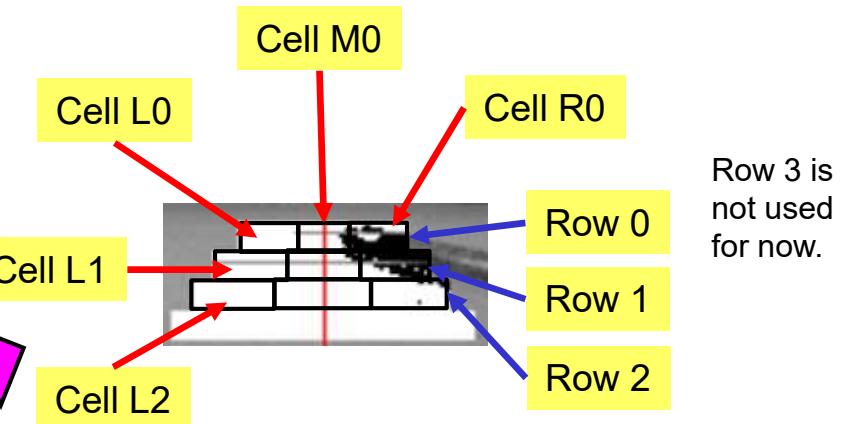
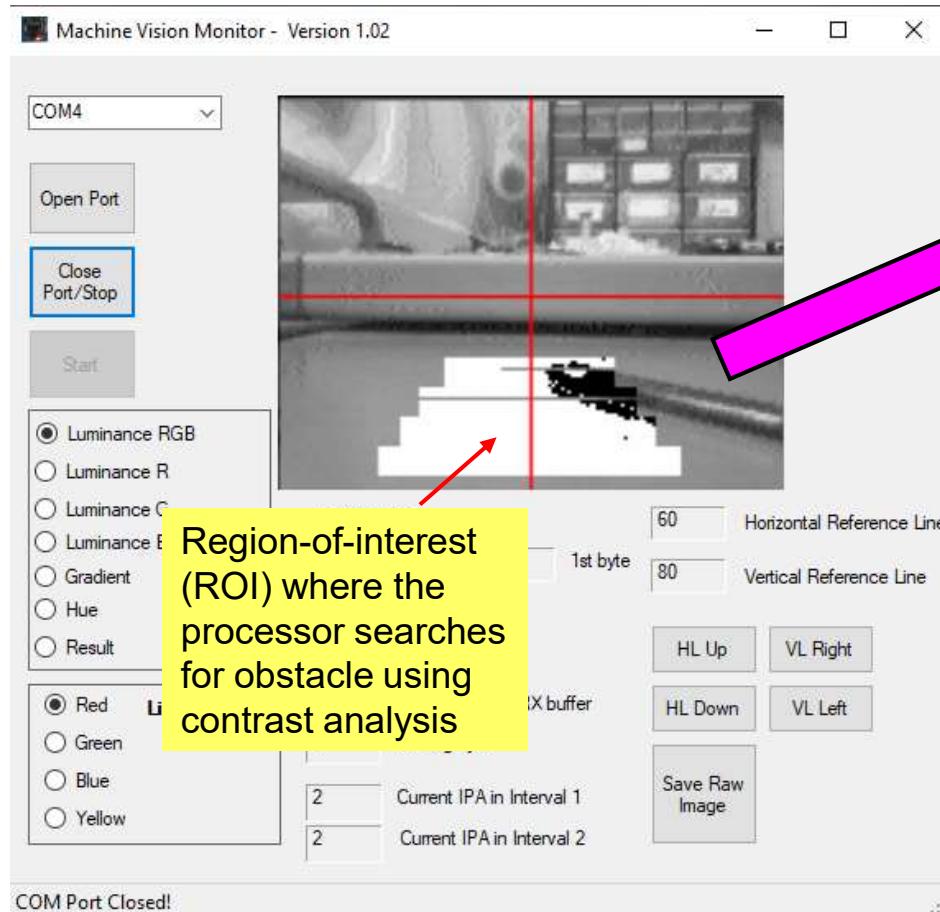
# Example 2 – The Assignment of IPAs to Intervals



- Each IPA only executes every 100 ms, thus the response time now slows down to 100 ms, however the up side is we get to run two different algorithms simultaneously.

# Interpreting the Results of IPA 2

- When IPA 2 is activated:



Whenever the number of black pixels exceed a pre-determined threshold in a cell, an obstacle is deemed present. Thus, for this example, cells R0, M0 and R1 contains obstacle and the bits corresponding to these cells will be set to '1'. The following result packet will be transmitted via UART 1 from MVM V1.5C:

Byte 1 = 2 ID

Byte 2 = 0b00000000 Row 2

Byte 3 = 0b00000001 Row 1

Byte 4 = 0b00000011 Row 0

L M R

---

# **Connection to TFT LCD Display**

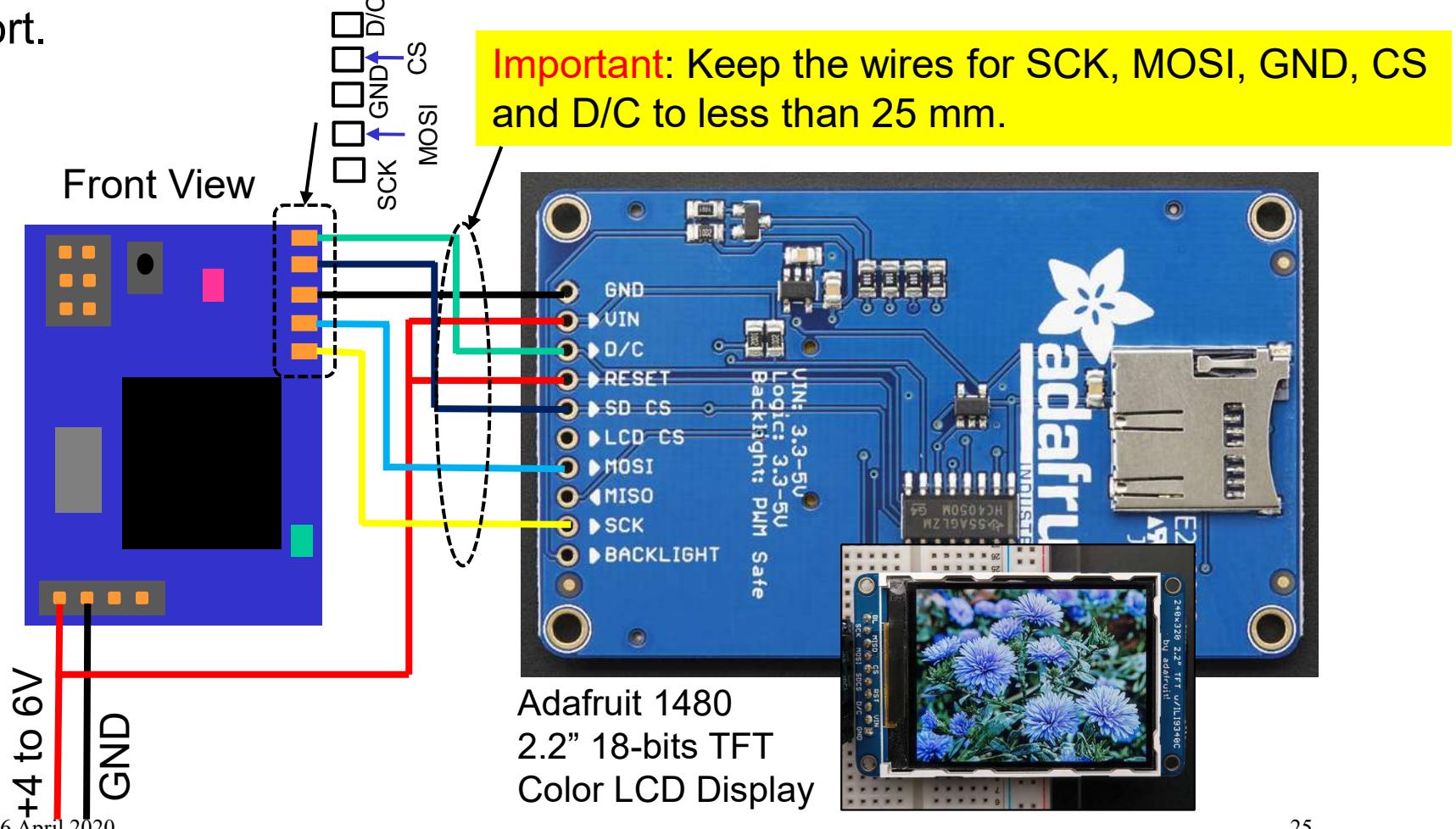
# Support for TFT LCD Display

---

- Presently only ILI9341 TFT LCD controller in 4-wires serial communication mode is supported. The ILI9341 TFT LCD controller support a resolution of 320x240 pixels (i.e. QVGA).
- So LCD display using this controller can be connected to the MVM V1.5C.
- Examples of compatible TFT Displays are products from Adafruit such as product ID 2478, 1770, 1743 and 1480.
- Due to limitation on the micro-controller bandwidth, the firmware can only support either streaming of image to PC (via serial port) or streaming of image to LCD, not both. As default the firmware pre-programmed into the micro-controller is the former. Thus if it is desire to interface the MVM V1.5C streaming output to TFT LCD display, another version of the firmware needs to be compiled and loaded into the micro-controller. See next section “Compiling and building your own firmware for MVM V1.5C”.

# Connection Diagram

- As the serial signals runs at 75 MHz, it is important to keep the wires short.



---

# **Compiling and Building Your Own Firmware for MVM V1.5C**

# Introduction 1

---

- The source codes for the sample firmware is a simplified version of the application pre-loaded into the MVM V1.5C micro-controller.
- The codes for IPA 1 is provided with the sample firmware and if the micro-controller is programmed with the sample firmware hex output, the micro-controller will run IPA 1 continuously at 20 fps upon power up.

# Introduction 2

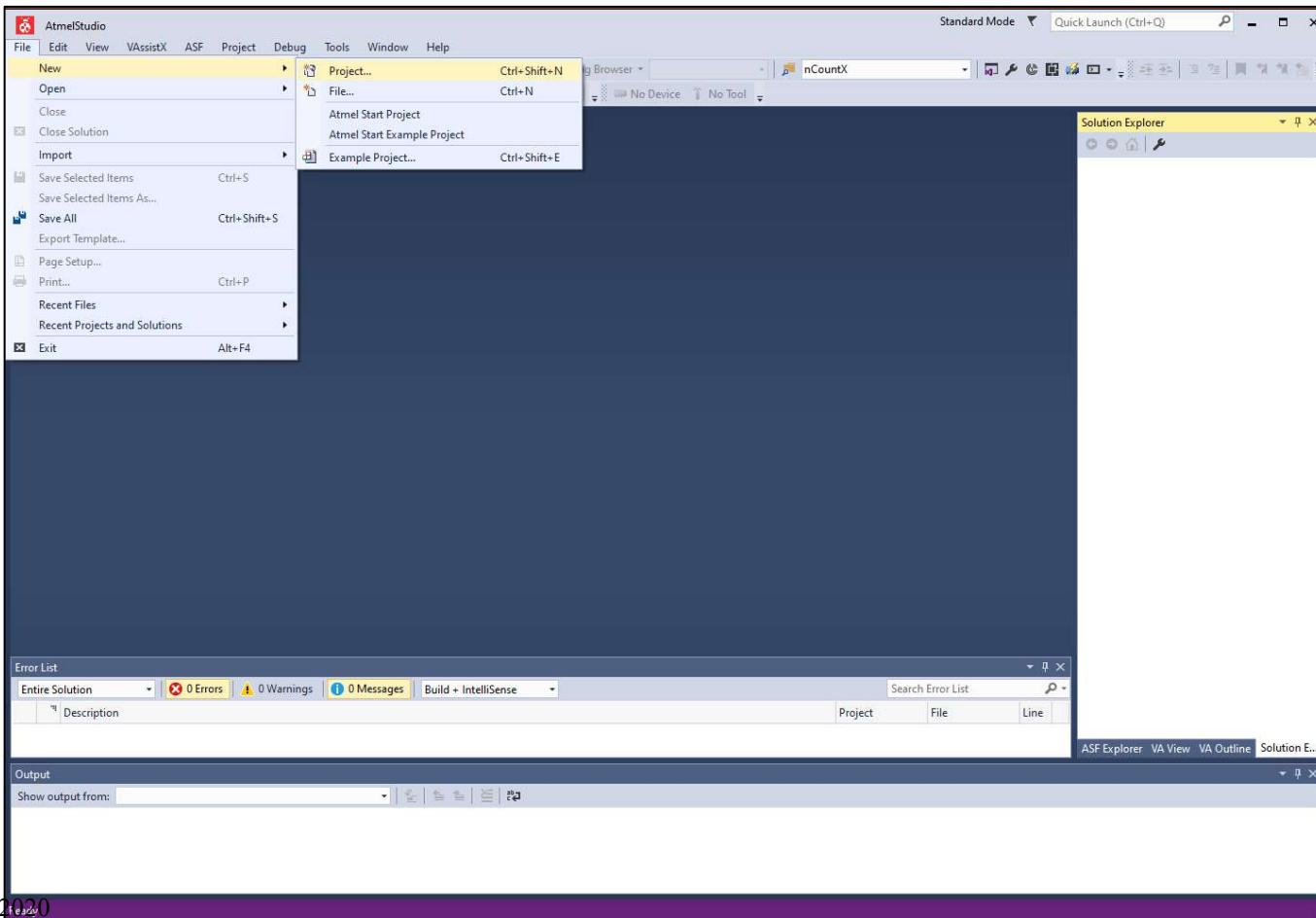
- Clone the MVM\_V1\_5C folder from  
[https://github.com/fabiankung/MVM\\_V1\\_5C](https://github.com/fabiankung/MVM_V1_5C)

MVM_Arduino_Example	Arduino sketch for Example 3 & 4	Original firmware in hex format
MVM_Miscellaneous/Scilab	Add files via upload	
MVM_Original_Hex_File_R0.54	Latest firmware, to be used with latest PC software.	PC Application and Source codes for MV Monitor software
MVM_PC_Monitor_Software	Delete MV_Monitor.exe	
MVM_Sample_Firmware_R0.9	Minor update on transmission of hue information	Sample firmware in C for MVM V1.5C, stream image to PC, <b>pre-programmed into unit</b>
MVM_Sample_Firmware_R0.95_CNN	Add files via upload	
MVM_Sample_Firmware_R0.9LCD	Create Readme	Sample firmware in C for MVM V1.5C (stream Image to TFT LCD)
LICENSE	Create LICENSE	
MVM_TensorflowCNNModel_June202...	Add files via upload	
MVM_V1_5C_QuickStartGuide_V0_96...	Latest quick start guide	
README.md	Update README.md	

- “MVM\_Sample\_Firmware” contains all the drivers files and IPA 1 routines. You can use this to build your own custom applications.
- “MVM\_PC\_Monitor\_Software” contains the Visual Studio template to build up the Machine Vision Monitor software in Visual Basic .NET.

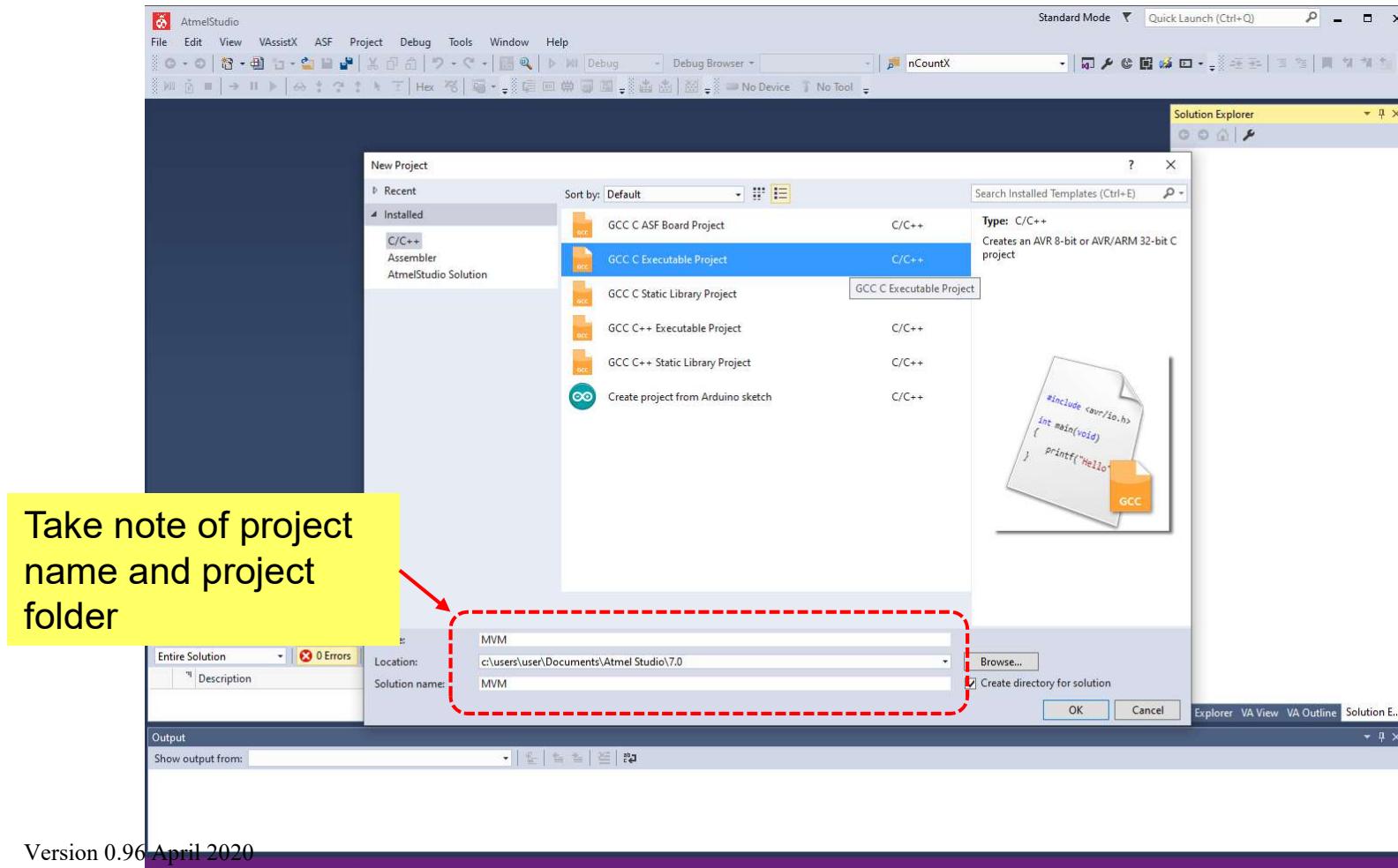
# Setting Up An Atmel Studio Project 1

- Start a new project in Atmel Studio 7.



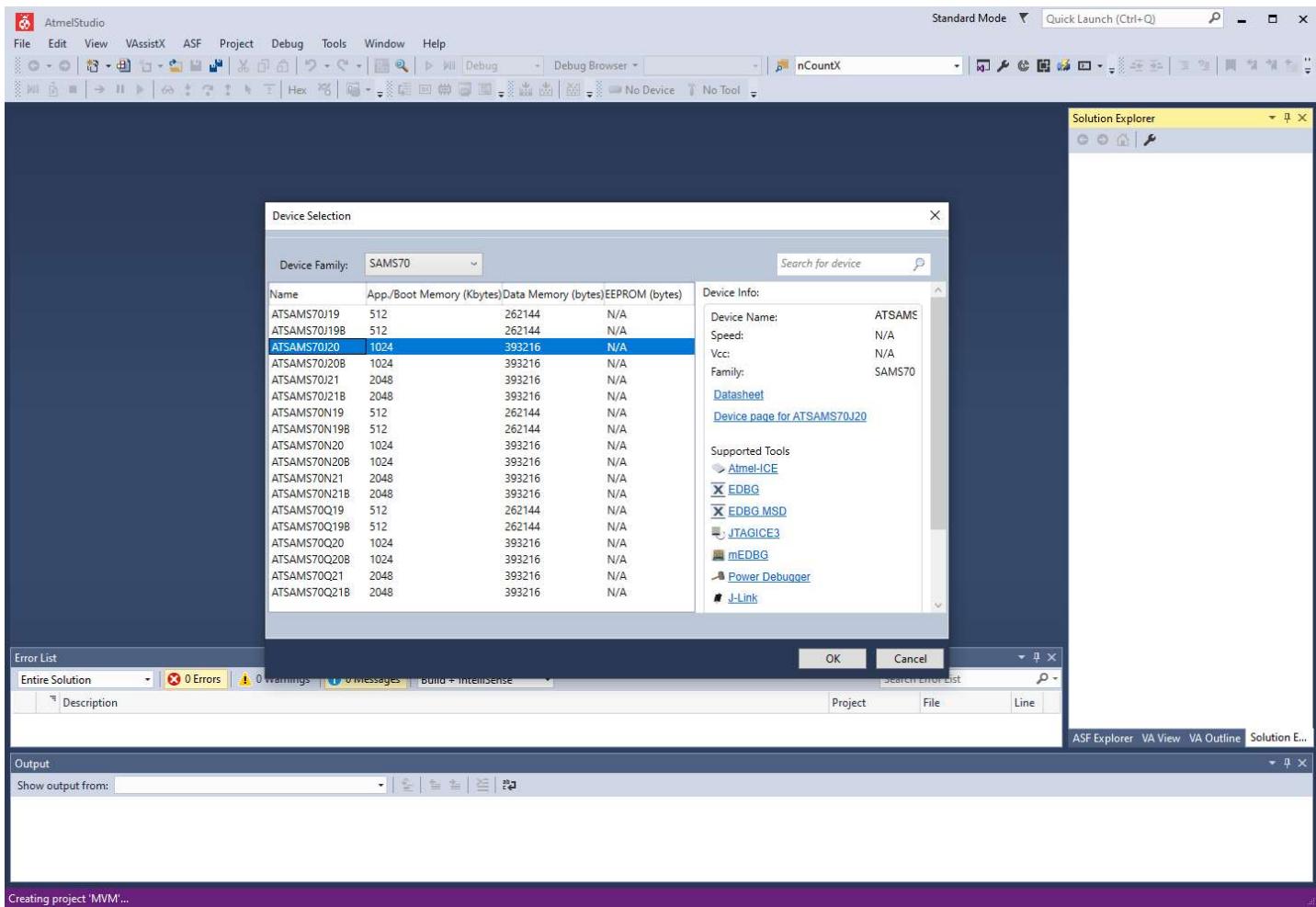
# Setting Up An Atmel Studio Project 2

- Create a GCC C executable project.



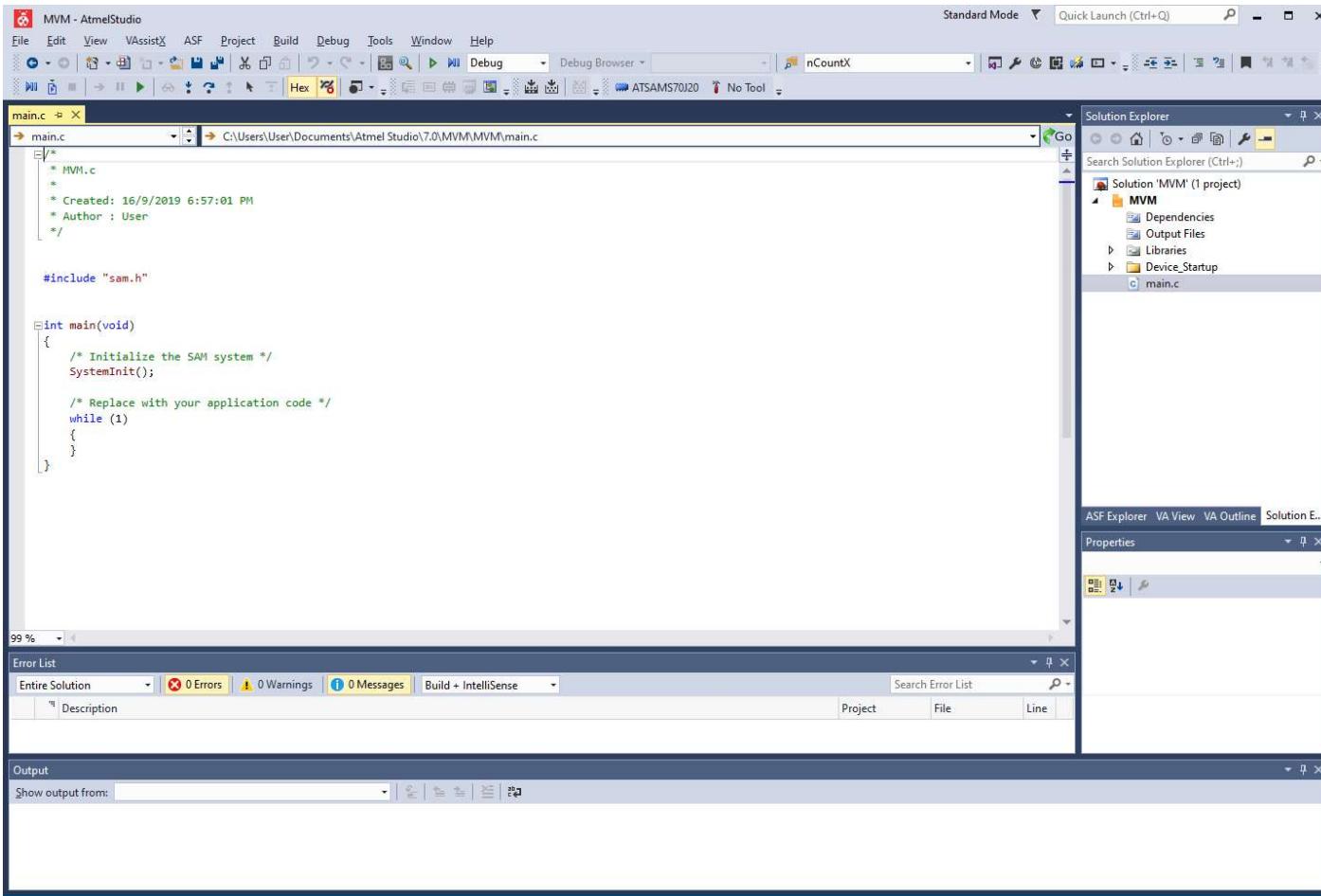
# Setting Up An Atmel Studio Project 3

- Select the correct device.



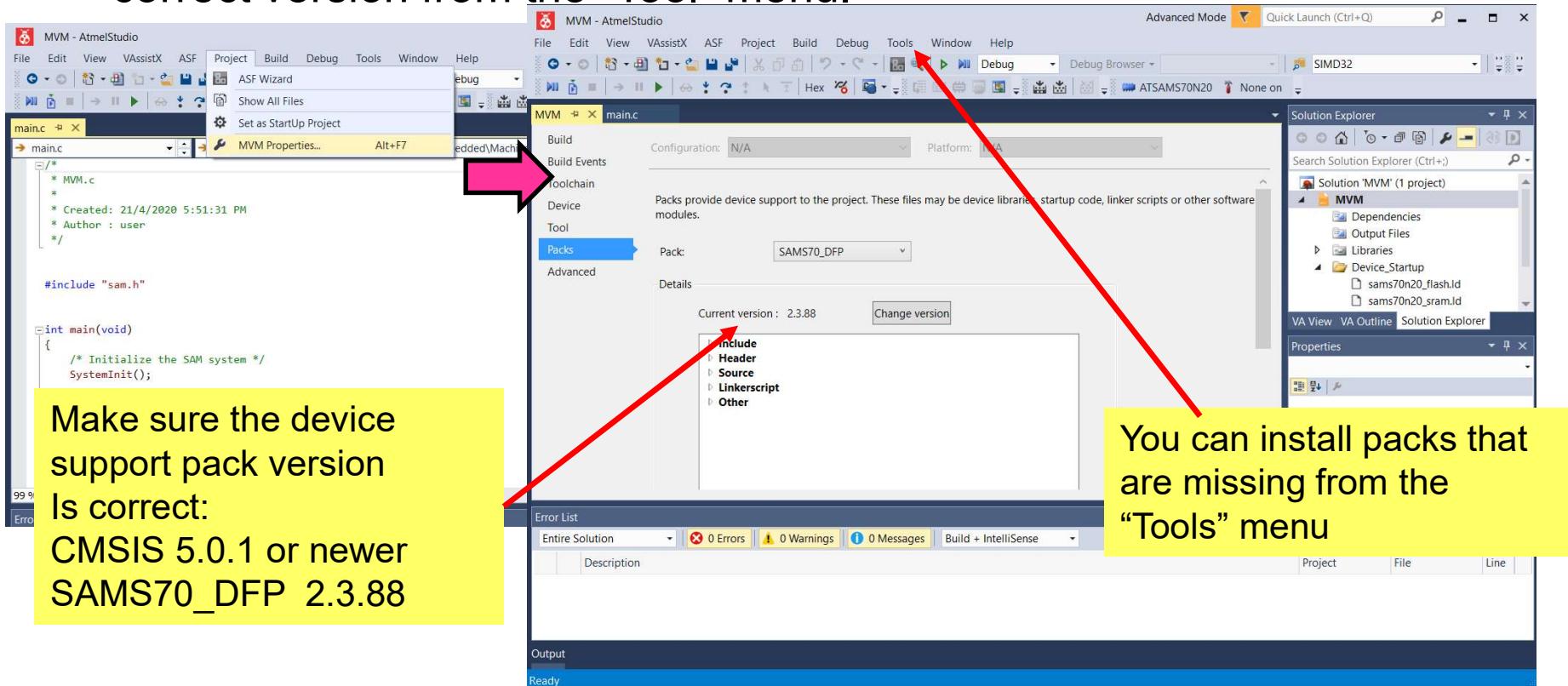
# Setting Up An Atmel Studio Project 4

- A project with a default “main.c” file will be created.



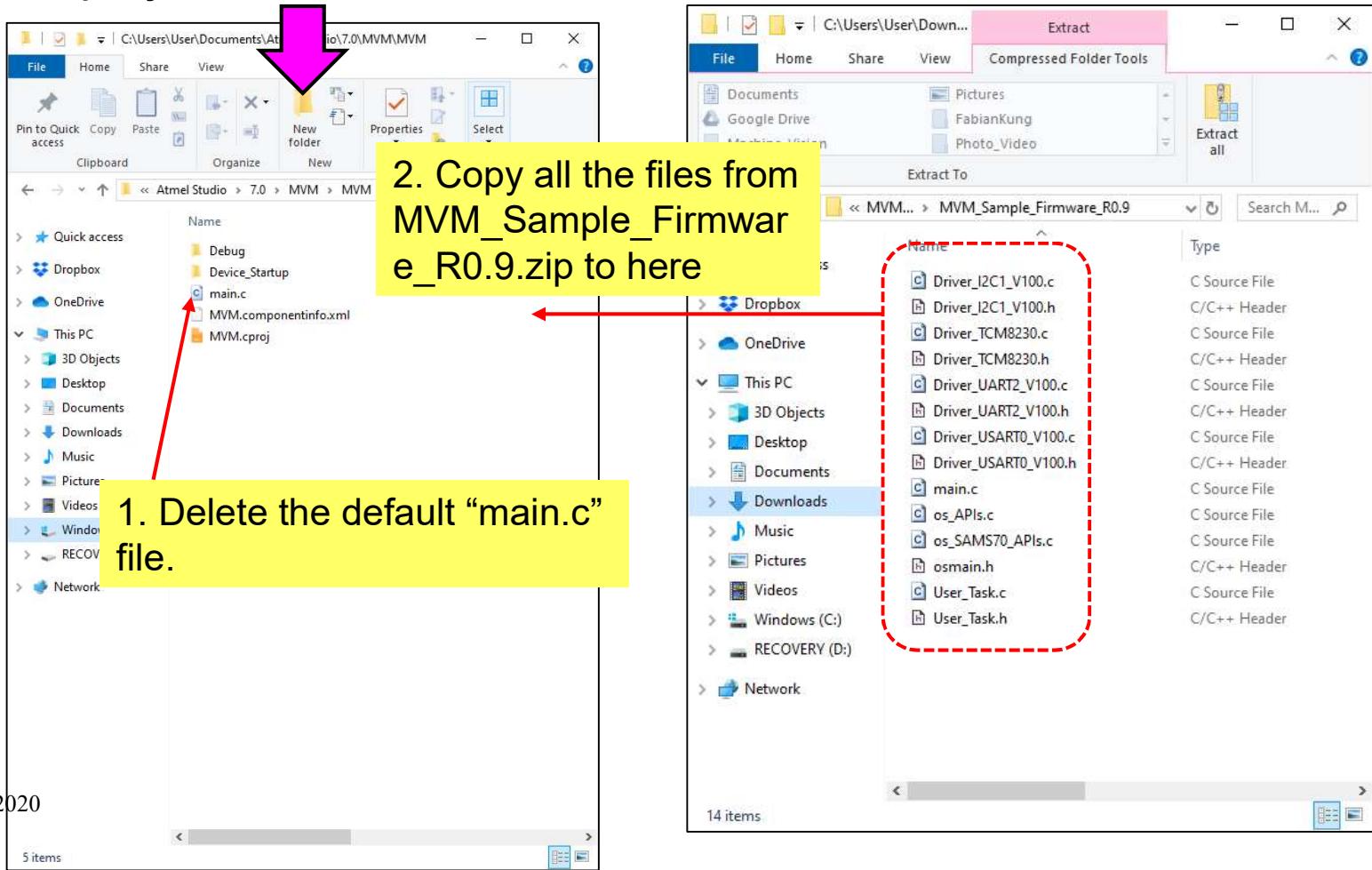
# Setting Up An Atmel Studio Project 5

- Now view the attributes of the project, make sure the CMSIS and device support package version is correct. Else you can install the correct version from the “Tool” menu.



# Setting Up An Atmel Studio Project 6

- Now close Atmel Studio 7.
- Go to the project folder.



# Setting Up An Atmel Studio Project 7

- Now reopen Atmel Studio 7. The new “main.c” file will be reflected window.

The screenshot shows the Atmel Studio 7 interface with the following details:

- Solution Explorer:** Shows the project structure for 'MVM' with a single item: 'main.c'. A red arrow points to this item with the text "New ‘main.c’ file.".
- Properties:** Shows 'MVM Project Properties' and 'Misc' section with 'Project File' set to 'MVM.cproj'.
- Output:** Shows the output window which is currently empty.
- Error List:** Shows the error list with 0 Errors, 0 Warnings, and 0 Messages.
- Code Editor:** Displays the 'main.c' source code. The code initializes the SAMS70 chip, creates tasks for various drivers and processes, and includes a main loop with a SysTick check.

```
main.c // C:\Users\User\Documents\Atmel Studio\7.0\MVM\MVM\Driver_TCM8230.h
#include "User_Task.h"

int main(void)
{
    int ni = 0;

    SAMS70_Init();           // Custom initialization, see file "os_SAMS70_APIs.c". This will overwrites the
                            // initialization done in SystemInit();
    OSInit();                // Custom initialization: Initialize the RTOS.
    gnTaskCount = 0;          // Initialize task counter.

    // Initialize core OS processes.
    OSCreateTask(&gstrcTaskContext[gnTaskCount], OSProce1);           // Start main blinking LED process.

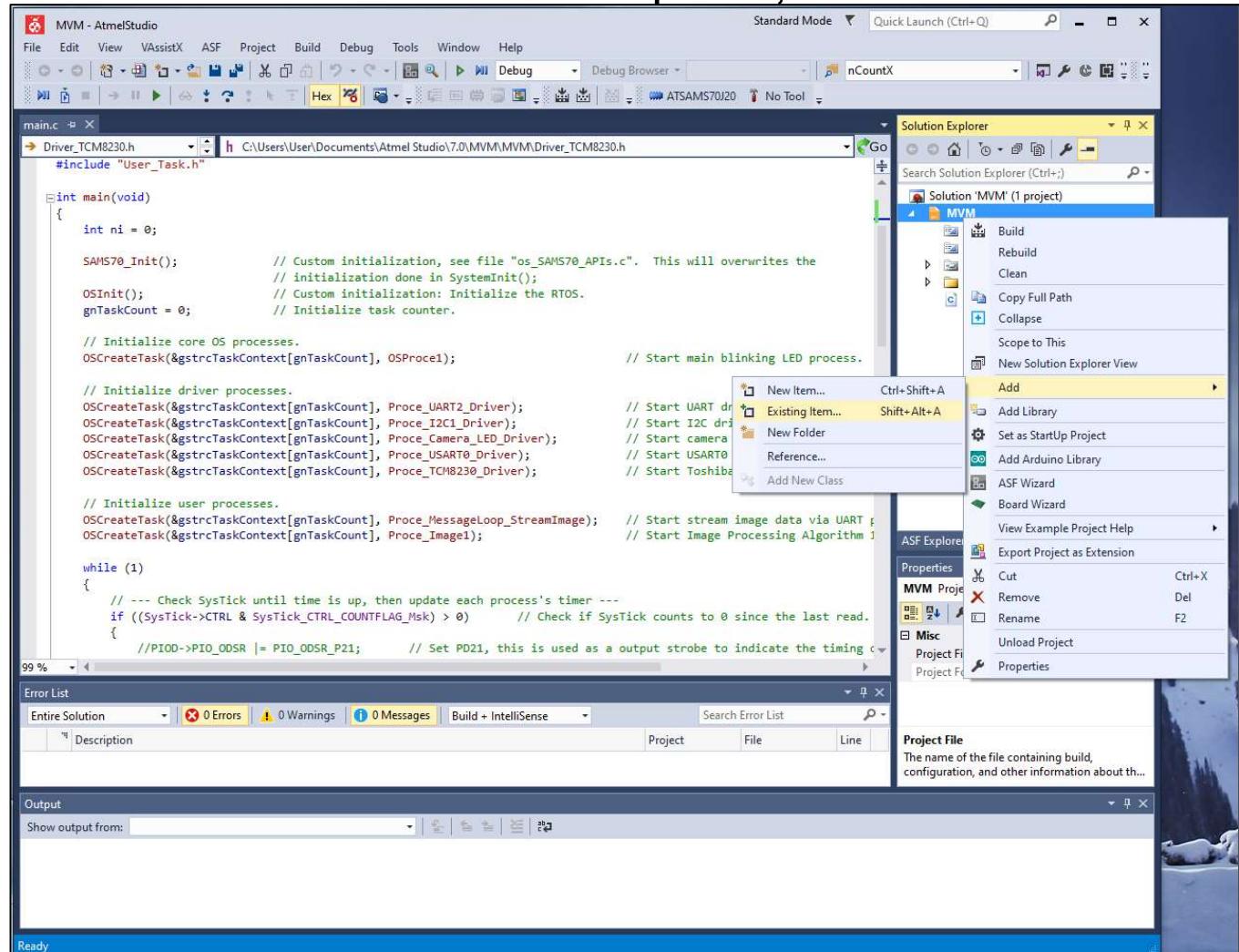
    // Initialize driver processes.
    OSCreateTask(&gstrcTaskContext[gnTaskCount], Proce_UART2_Driver);   // Start UART driver.
    OSCreateTask(&gstrcTaskContext[gnTaskCount], Proce_I2C1_Driver);     // Start I2C driver.
    OSCreateTask(&gstrcTaskContext[gnTaskCount], Proce_Camera_LED_Driver); // Start camera LED driver.
    OSCreateTask(&gstrcTaskContext[gnTaskCount], Proce_USART0_Driver);   // Start USART0 driver.
    OSCreateTask(&gstrcTaskContext[gnTaskCount], Proce_TCH8230_Driver);  // Start Toshiba TCH8230 camera driver

    // Initialize user processes.
    OSCreateTask(&gstrcTaskContext[gnTaskCount], Proce_MessageLoop_StreamImage); // Start stream image data via UART p
    OSCreateTask(&gstrcTaskContext[gnTaskCount], Proce_Image1);           // Start Image Processing Algorithm J

    while (1)
    {
        // --- Check SysTick until time is up, then update each process's timer ---
        if ((SysTick->CTRL & SysTick_CTRL_COUNTFLAG_Msk) > 0)           // Check if SysTick counts to 0 since the last read.
        {
            //PIO0->PIO_ODSR |= PIO_ODSR_P21;      // Set PD21, this is used as a output strobe to indicate the timing c
        }
    }
}
```

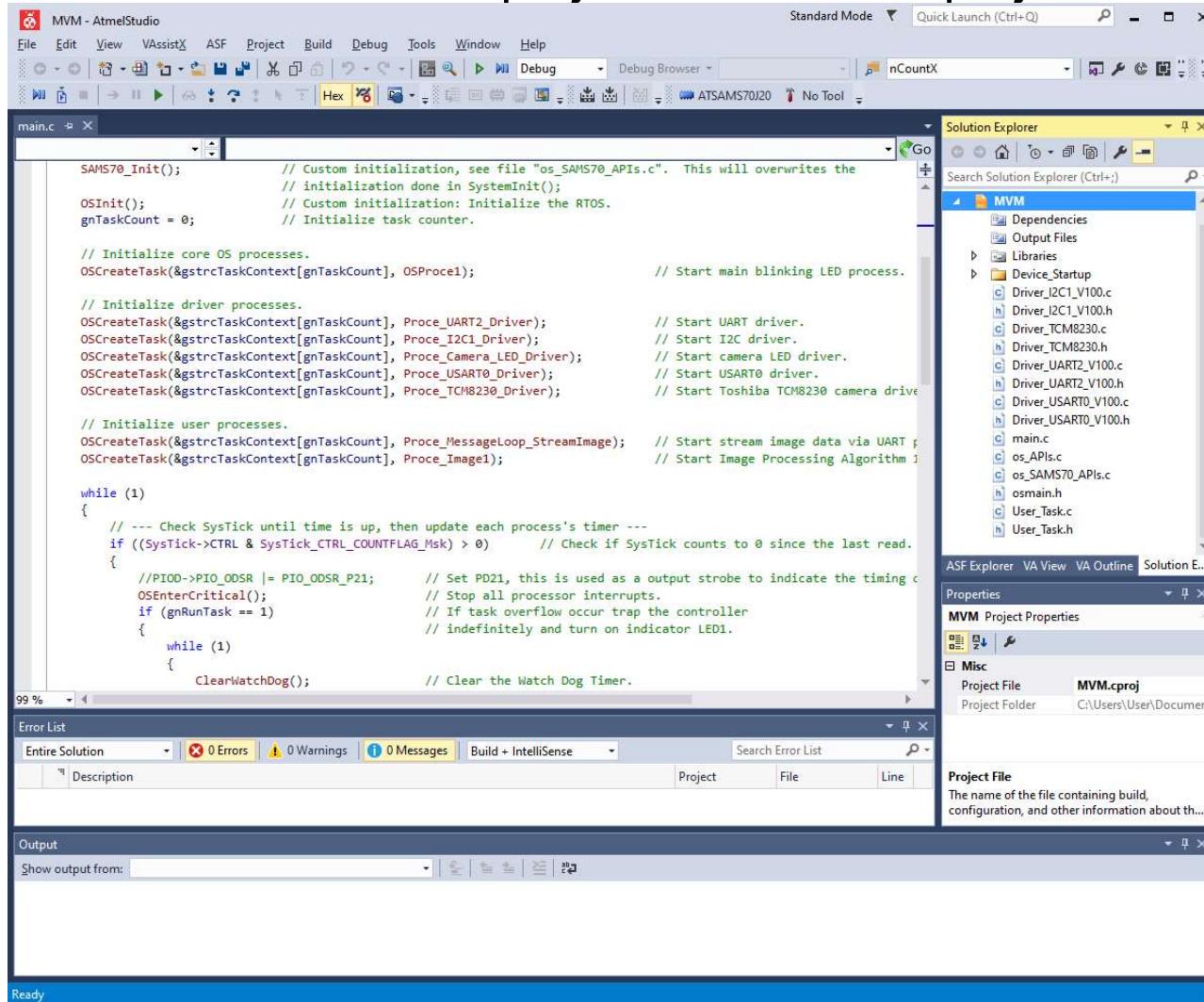
# Setting Up An Atmel Studio Project 8

- Right click the folder “MVM” in the Solution Explorer, and select Add Existing Item...



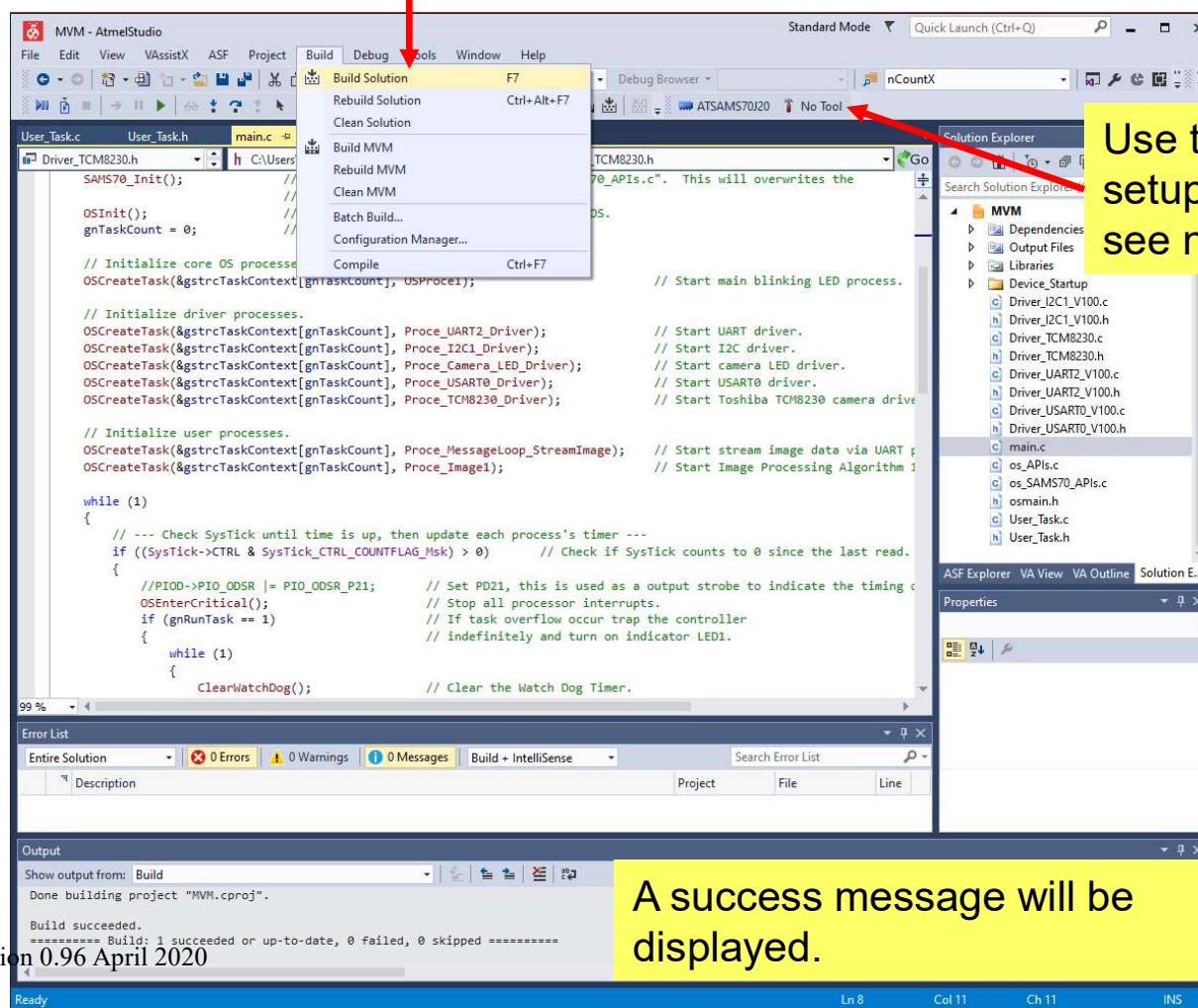
# Setting Up An Atmel Studio Project 9

- Add all the \*.c and \*.h files in the project folder to the project.



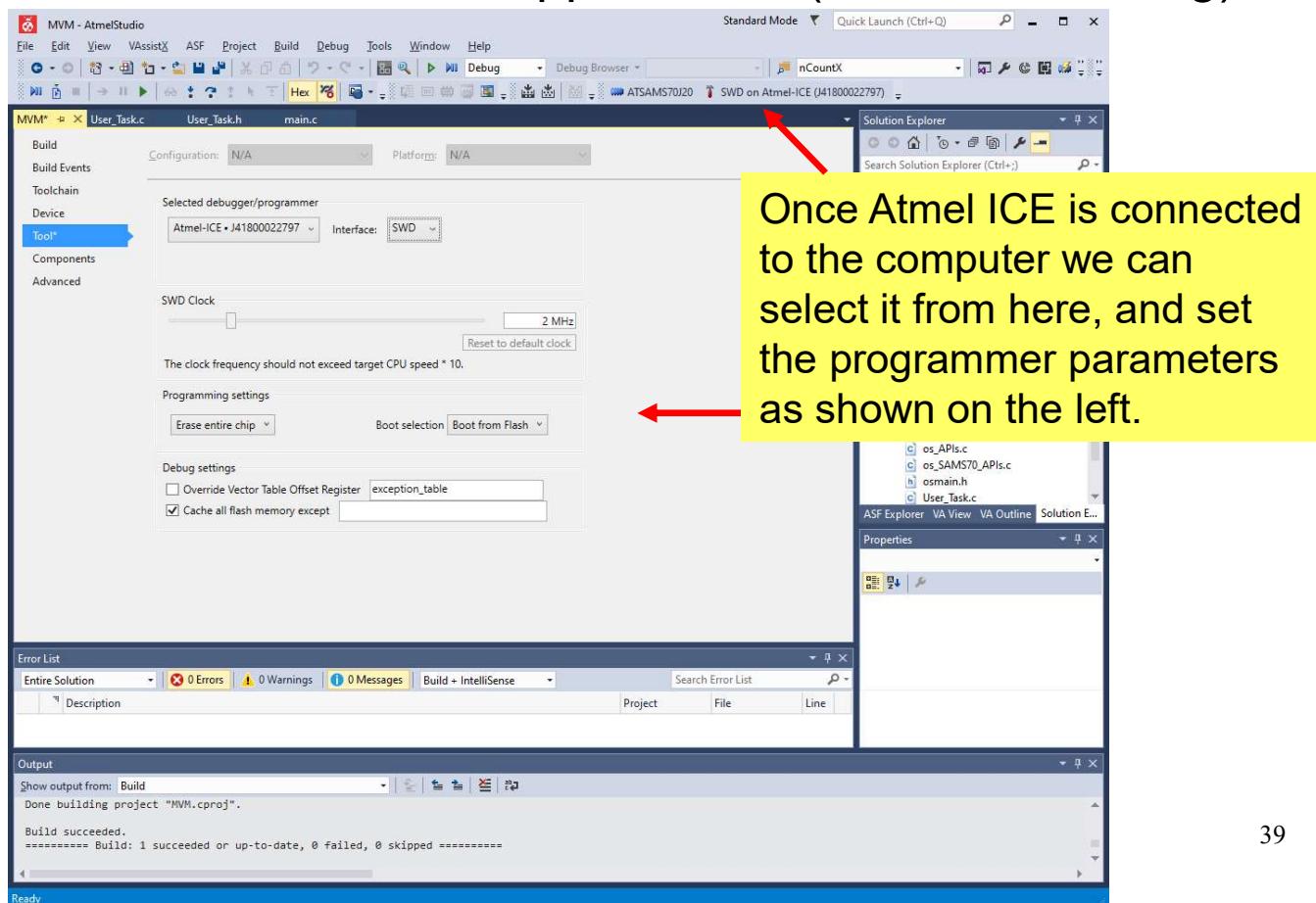
# Setting Up An Atmel Studio Project 10

- Now you can build or compile the project.



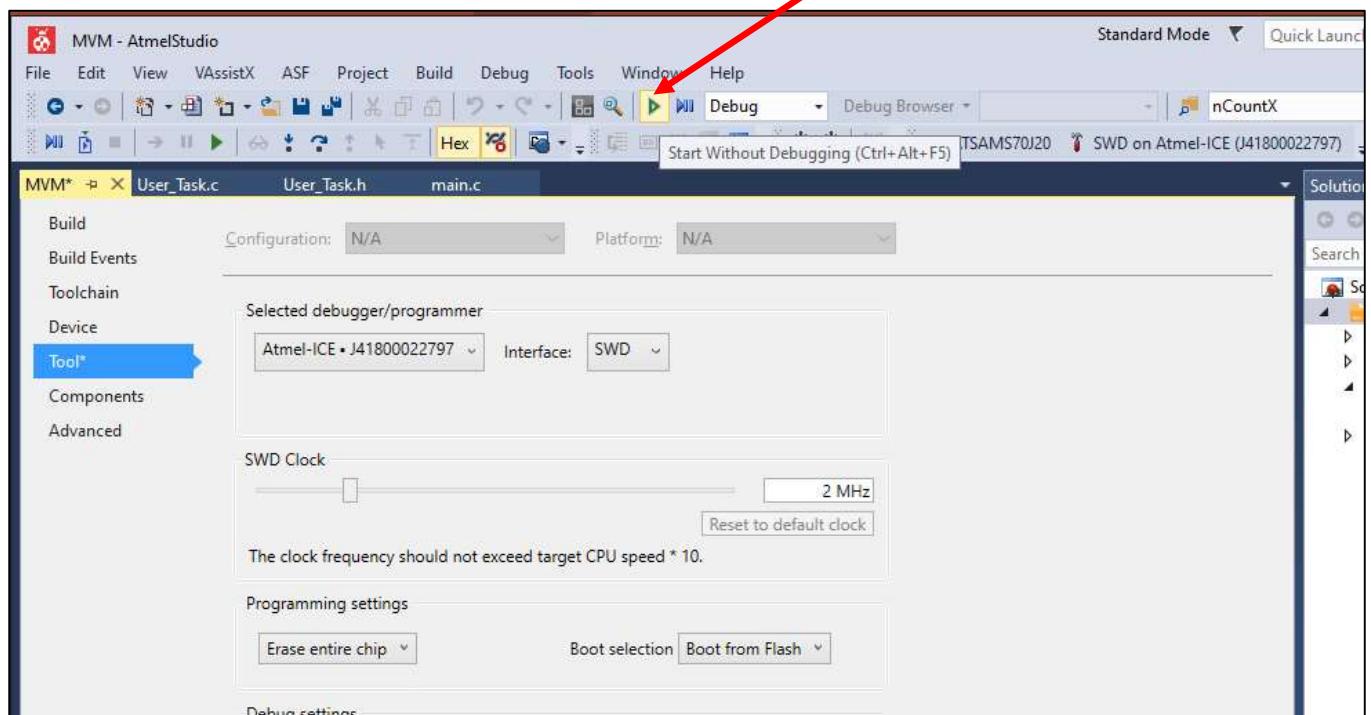
# Setting Up the Programming Tool – Atmel ICE

- Now you can load the firmware into the micro-controller with a suitable programmer. Here we are using Atmel ICE, but any programmer compatible with Atmel Studio 7 and support SWD (serial wire debug) mode is fine.



# Flashing the Micro-Controller 1

- Connect the MVM to Atmel ICE. Power up the MVM and click this button to program the flash memory.
- See **Appendix** on the pin assignment on the 2x3 ways receptacle that comes with Atmel ICE.



# Flashing the Micro-Controller 2

- Finally you need to setup the TCM (tightly coupled memory) size of Cortex M7 by setting the GPNVM (general purpose non-volatile memory) bits of SAMS70 as shown.

The screenshot shows the AtmelStudio interface with the 'Tool' tab selected. A red arrow points to the 'Device Programming' option in the 'Tools' menu. The main window displays the 'Atmel-ICE (J41800022797) - Device Programming' dialog. In the 'GPNVM Bits' section, two checkboxes are checked: 'GPNVMBITS.BOOT\_MODE' (Value: checked) and 'GPNVMBITS.TCM\_CONFIGURATION' (Value: 0x01). Below this, a register table shows 'GPNVMBITS' with a value of '0x0182'. A yellow callout box states: 'The Heartbeat LED of the MVM V1.5C should start blinking once all the GPNVM bits are programmed.' A red arrow points to the 'Program' button at the bottom right of the dialog. Another yellow callout box states: 'Hit the 'Program' button once the parameters are properly setup.' A red arrow points to the 'Program' button.

MVM - AtmelStudio

File Edit View VAssistX ASF Project Build Debug

User\_Task.c User\_Task.h main.c MVM

Build Build Events Toolchain Device Tool Components Advanced

Configuration: N/A

Selected debugger/programmer: Atmel-ICE • J41800022797

SWD Clock: 100000000 Hz (Reset to default clock)

The clock frequency should not exceed target CPU speed \* 10.

Programming settings: Erase entire chip, Boot selection: Boot from Flash

Debug settings: Override Vector Table Offset Register: exception\_table, Cache all flash memory except: [empty]

Atmel-ICE (J41800022797) - Device Programming

Tool Device Interface Device signature Target Voltage

Atmel-ICE ATSAMS70J20 SWD Apply 0xA1120C01 Read 3.3 V Read

Interface settings  
Tool information  
Device information  
Memories  
**GPNVM Bits**  
Lock bits  
Security

GPNVMBITS BOOT\_MODE Value: checked  
GPNVMBITS TCM\_CONFIGURATION Value: 0x01

Register Value  
GPNVMBITS 0x0182

Auto read Verify after programming

Starting operation read registers  
Reading register GPNVMBITS...OK  
Read registers...OK

Read registers...OK

Copy to clipboard Program Verify Read Close

The Heartbeat LED of the MVM V1.5C should start blinking once all the GPNVM bits are programmed.

Hit the 'Program' button once the parameters are properly setup.

Version 0.96 April 2020

41

# Coding Your Own Routines

---

- The source files “**User\_Task.c**” and “**User\_Task.h**” contains the routines and declarations for **image processing task 1** that search for the brightest region in an image.
- Use this as the basis to add on your own routines. Do remember to use the state machine approach to code your tasks, and keep the total execution time for all tasks within 1 system ticks!
- For more information on the round-robin scheduler and basic structure of the C codes for ARM Cortex-M see  
<https://fkeng.blogspot.com/2016/02/atmel-arm-cortex-m4-microcontroller.html>

---

# **Compiling and Building the Machine Vision Monitor Software**

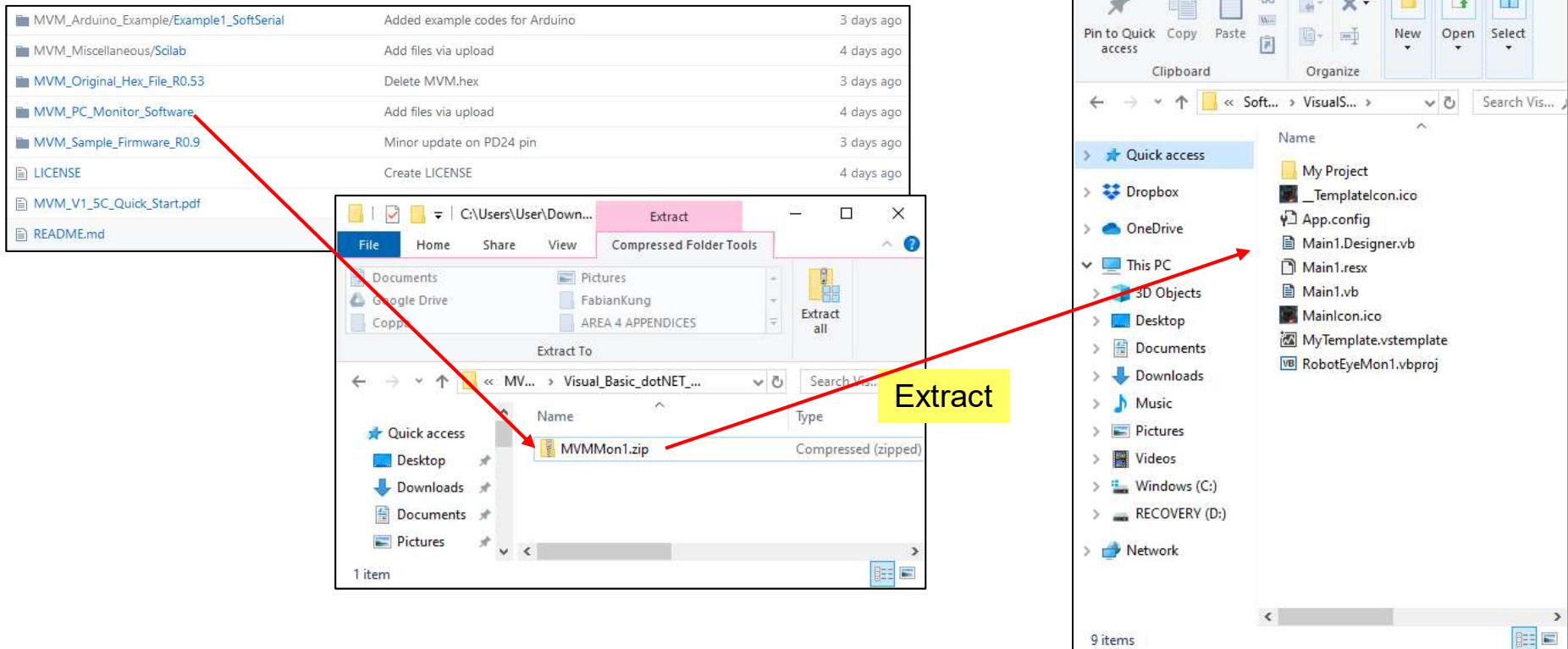
# Introduction

---

- The PC application (\*.exe) to observe the image frames captured by the MVM and the corresponding source codes are also provided.
- If needed, you can rebuild the application using Visual Studio Community version and customize the software features.
- The following slides show how to setup the Visual Studio project from the source codes provided.

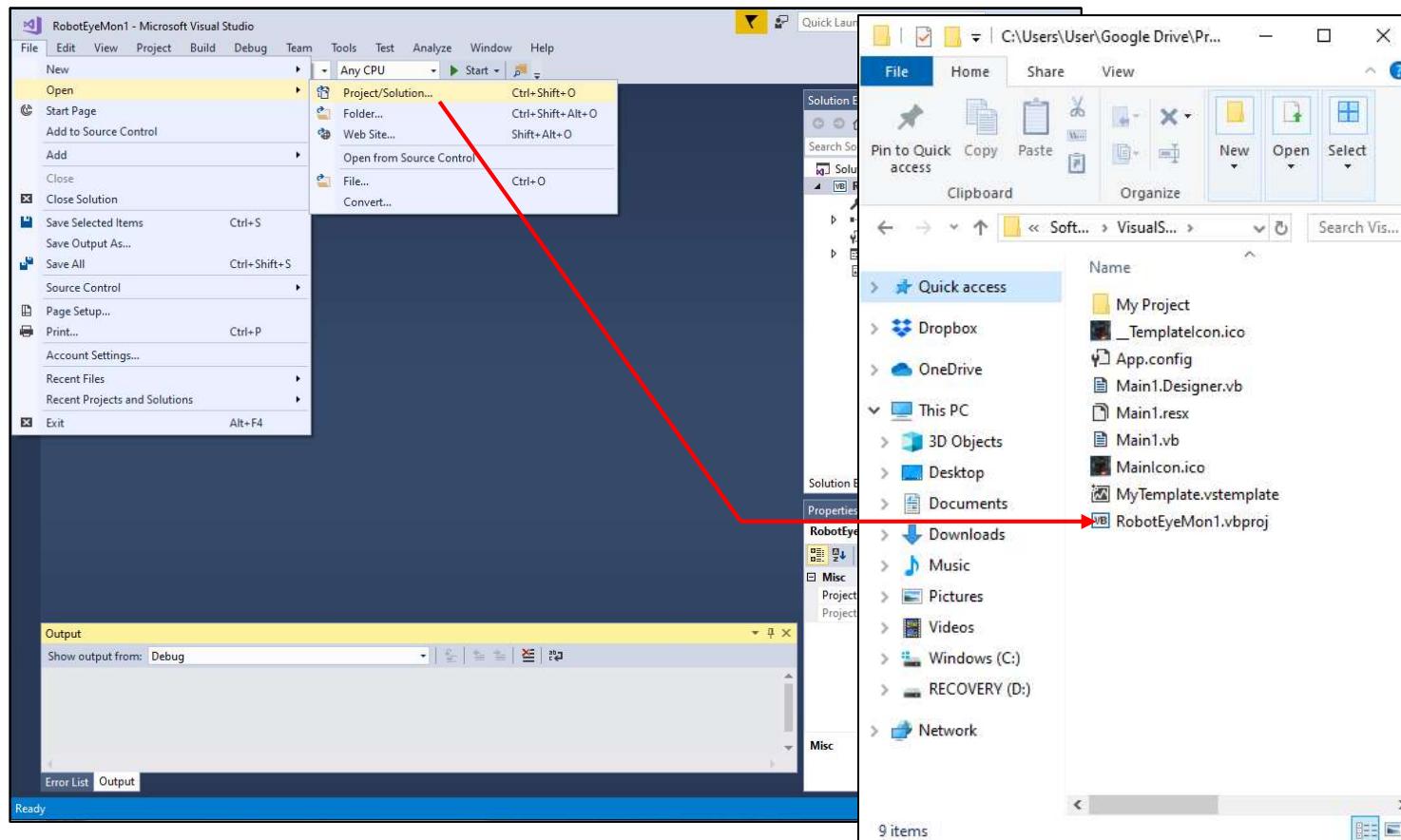
# Setting Up Visual Studio Project 1

- In the folder “MVM\_PC\_Monitor\_Software” look for the file MVMMon1.zip.
- Decompress the file into a suitable project folder.



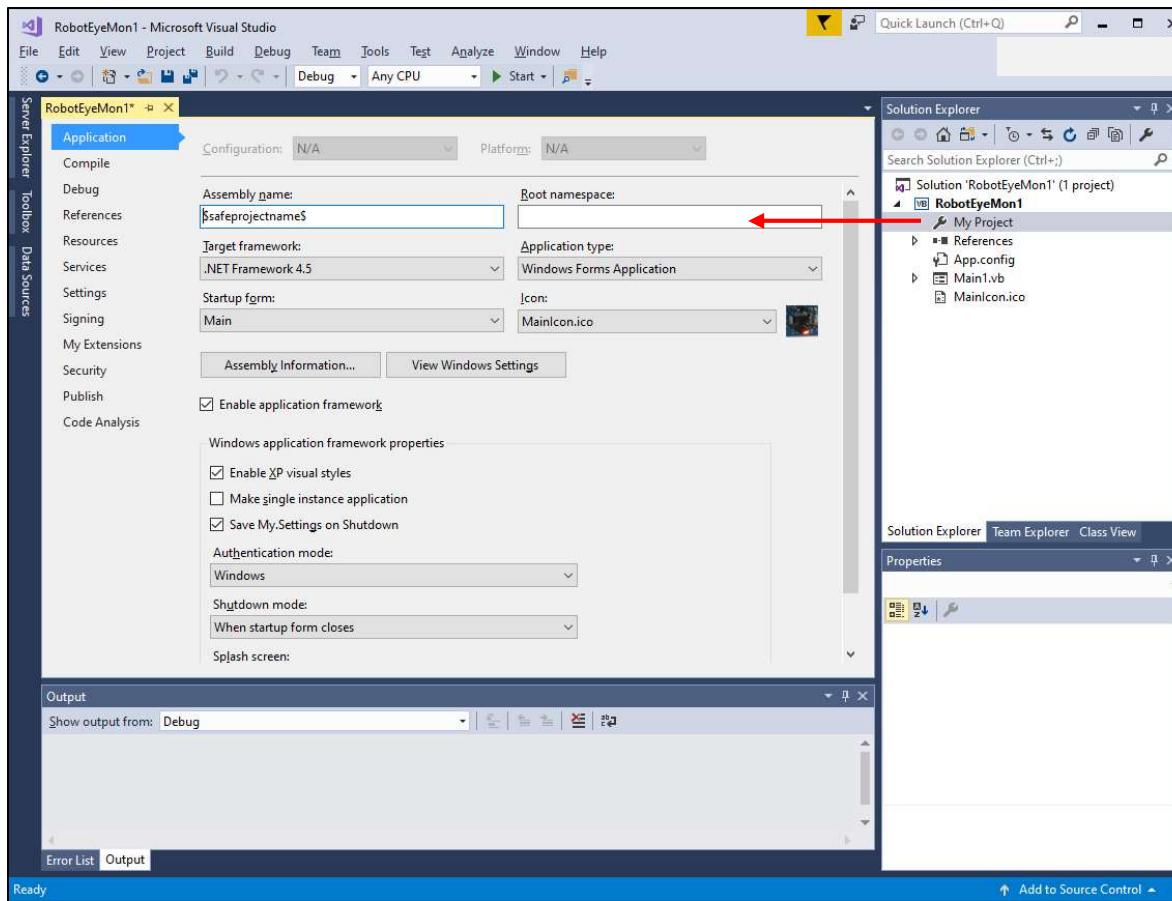
# Setting Up Visual Studio Project 2

- Open Visual Studio, and open the VB project (\*.vbproj) as shown.



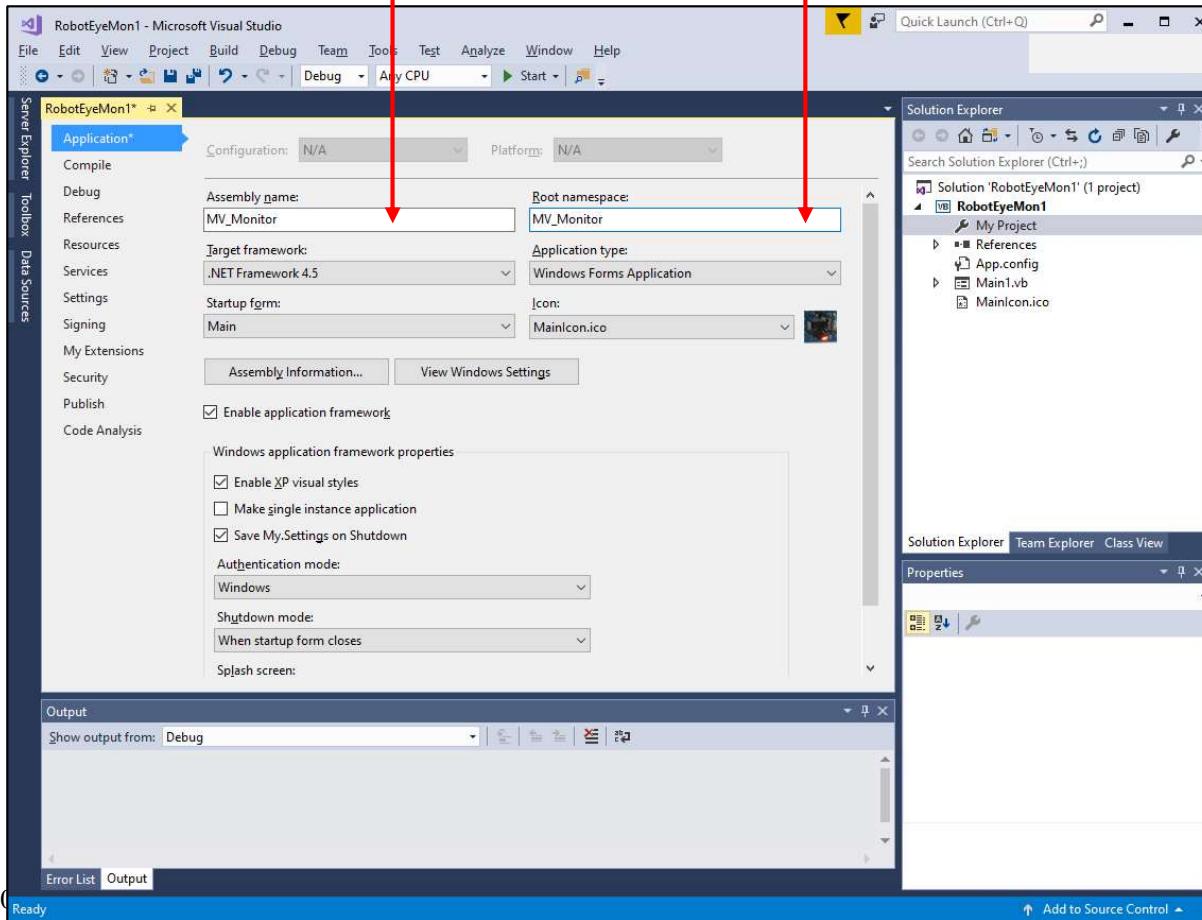
# Setting Up Visual Studio Project 3

- Double-click the MyProject icon to bring up the project setting.



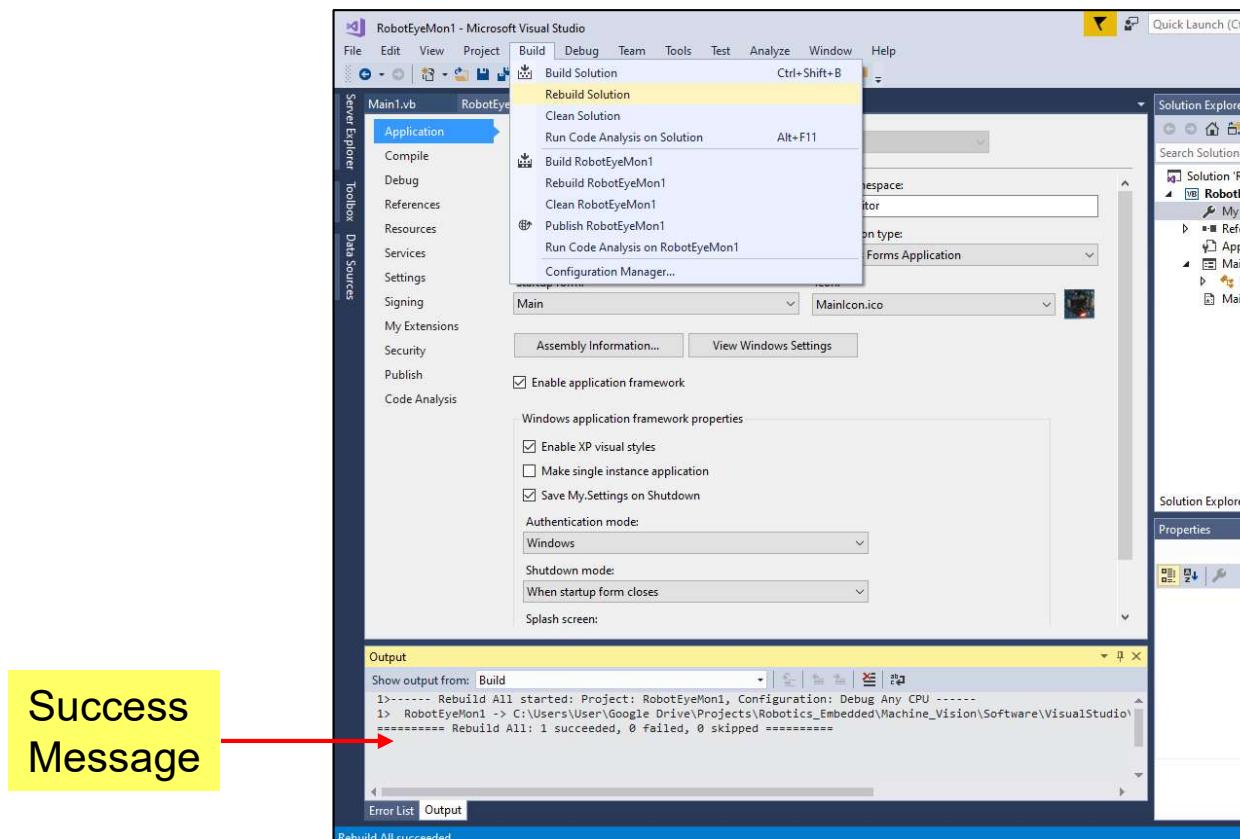
# Setting Up Visual Studio Project 4

- Type in the Assembly Name and Root Namespace as shown.



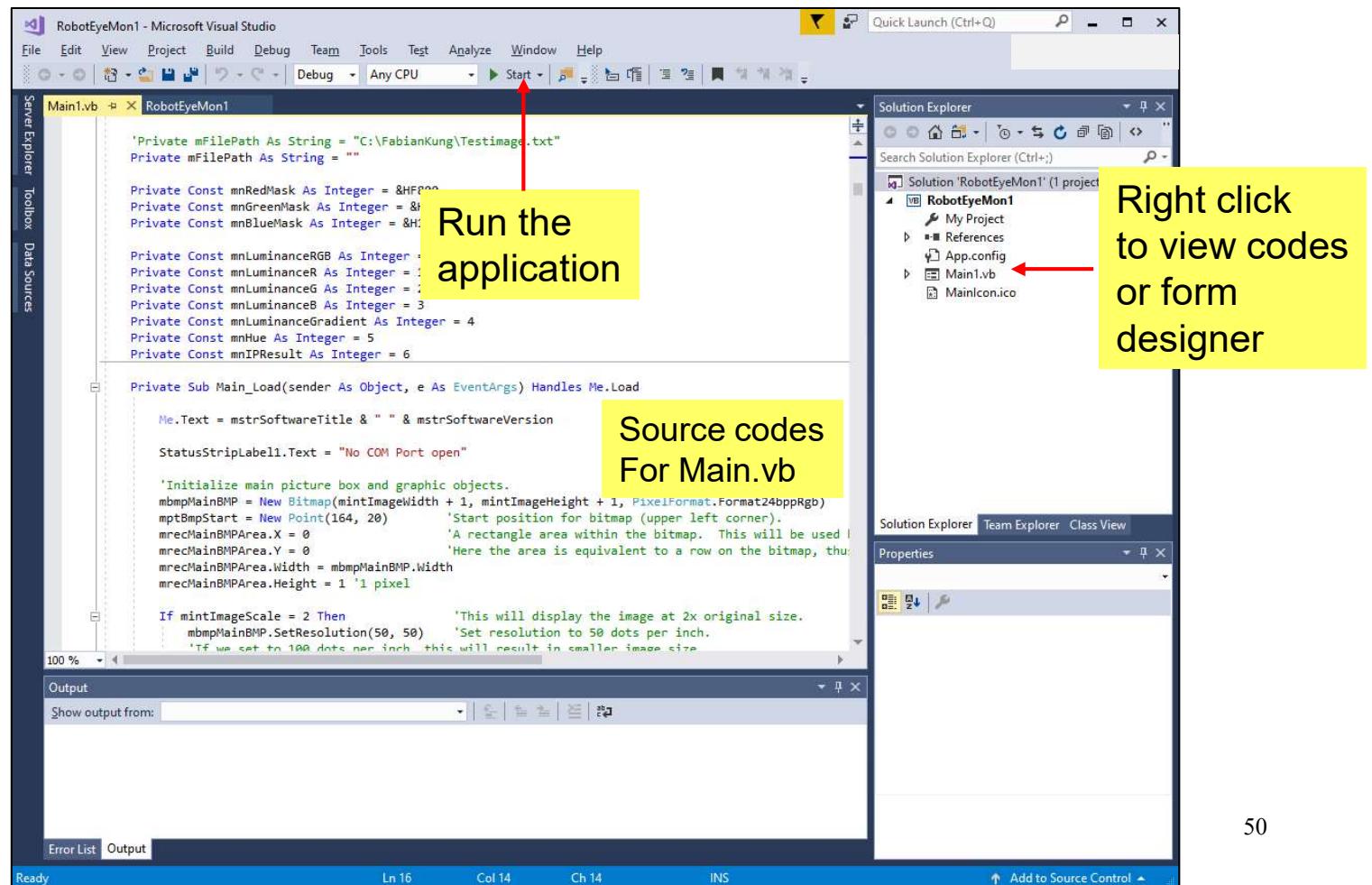
# Setting Up Visual Studio Project 5

- Now rebuild the project as shown and you should get a success message in the Output window.



# Setting Up Visual Studio Project 6

- You can now run the application, view/edit the source code and the main window form.

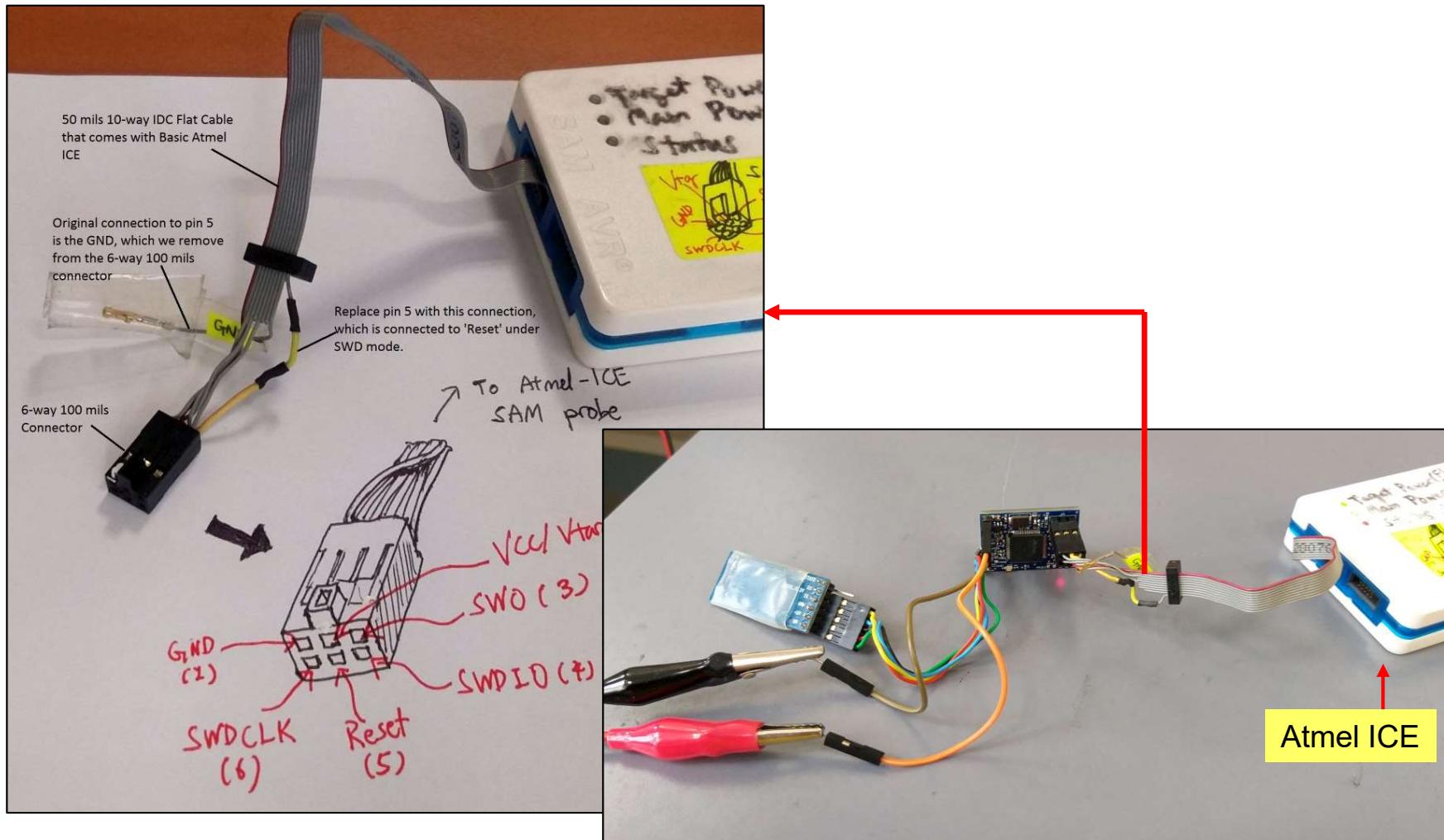


---

# APPENDIX

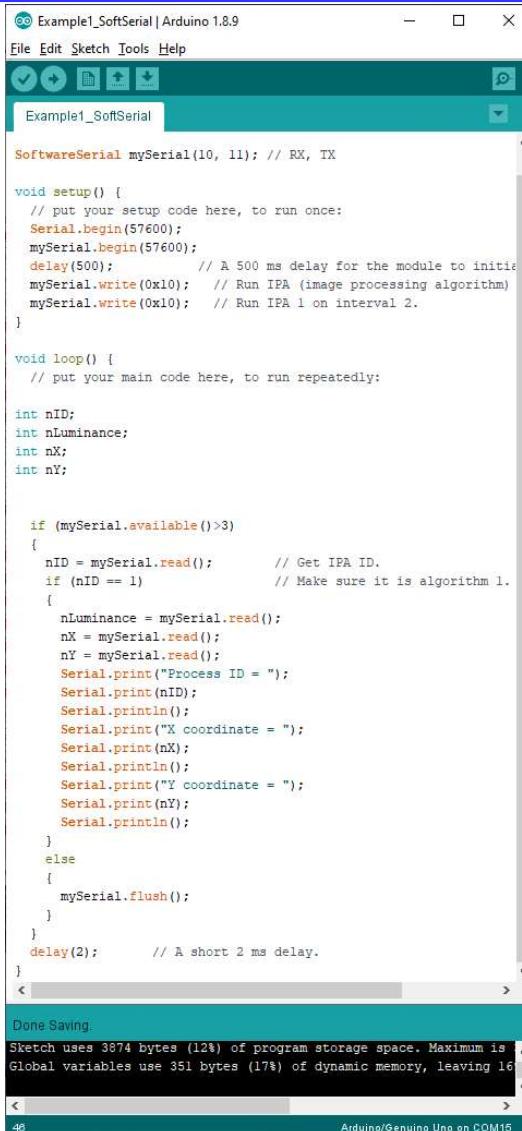
Programmer Connection, Examples and File Export

# Connecting Atmel ICE to MVM V1.5C



# Example 1 - Using MVM with Arduino Uno and SoftwareSerial Library

In this code we use SoftwareSerial port to communicate with MVM V1.5C, while the hardware serial is used in conjunction with Serial Terminal for debugging.



The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** Example1\_SoftSerial | Arduino 1.8.9
- Menu Bar:** File Edit Sketch Tools Help
- Toolbar:** Standard toolbar icons for Open, Save, Print, etc.
- Sketch Name:** Example1\_SoftSerial
- Code Area:** Contains the following Arduino sketch code:

```
SoftwareSerial mySerial(10, 11); // RX, TX

void setup() {
    // put your setup code here, to run once:
    Serial.begin(57600);
    mySerial.begin(57600);
    delay(500);          // A 500 ms delay for the module to initialize
    mySerial.write(0x10); // Run IPA (image processing algorithm)
    mySerial.write(0x10); // Run IPA 1 on interval 2.
}

void loop() {
    // put your main code here, to run repeatedly:

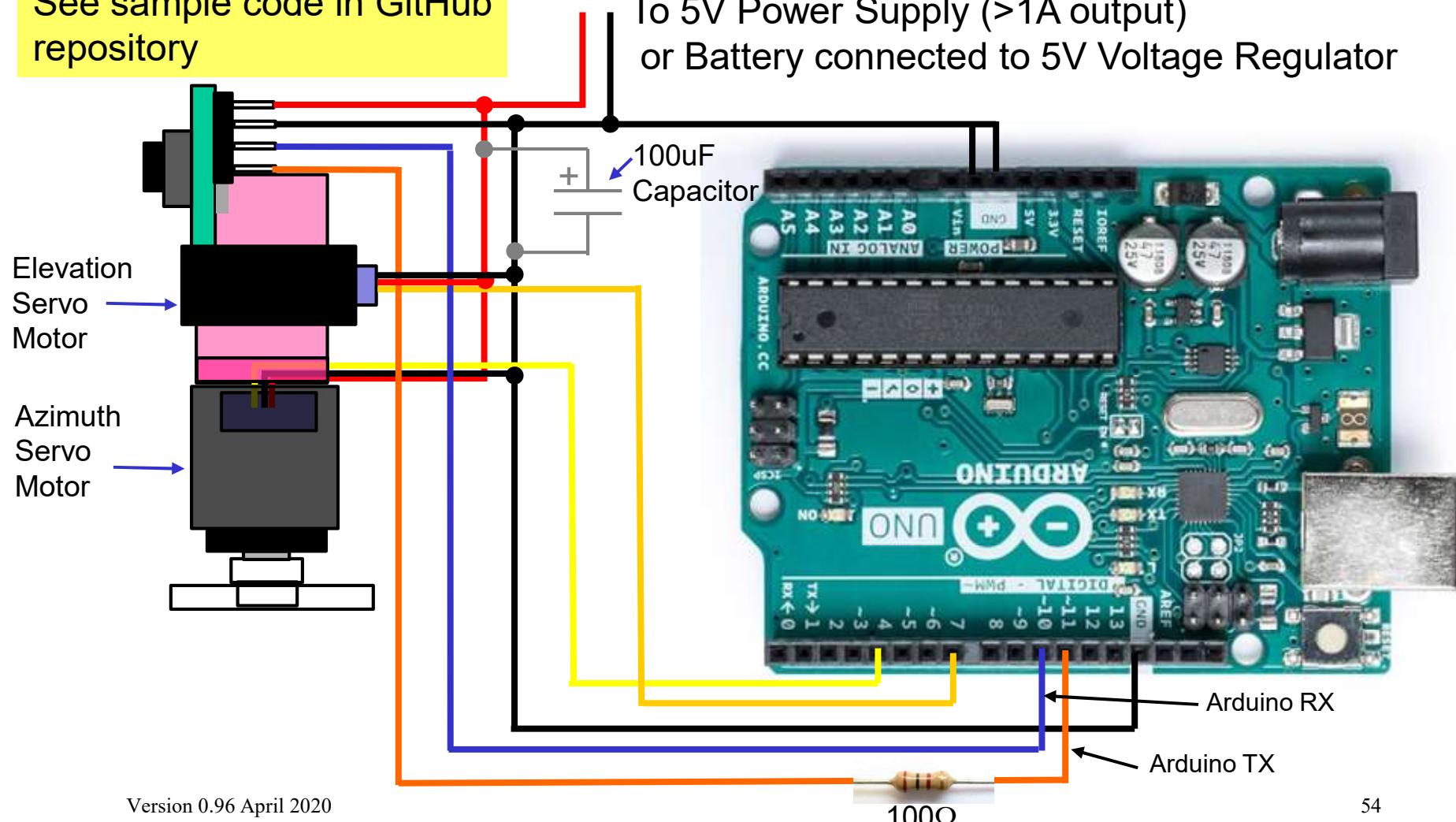
    int nID;
    int nLuminance;
    int nX;
    int nY;

    if (mySerial.available()>3)
    {
        nID = mySerial.read();           // Get IPA ID.
        if (nID == 1)                  // Make sure it is algorithm 1.
        {
            nLuminance = mySerial.read();
            nX = mySerial.read();
            nY = mySerial.read();
            Serial.print("Process ID = ");
            Serial.print(nID);
            Serial.println();
            Serial.print("X coordinate = ");
            Serial.print(nX);
            Serial.println();
            Serial.print("Y coordinate = ");
            Serial.print(nY);
            Serial.println();
        }
        else
        {
            mySerial.flush();
        }
        delay(2);           // A short 2 ms delay.
    }
}
```
- Status Bar:** Done Saving.  
Sketch uses 3874 bytes (12%) of program storage space. Maximum is 32256 bytes.  
Global variables use 351 bytes (17%) of dynamic memory, leaving 16148 bytes free.
- Bottom Status:** 48 Arduino/Genuino Uno on COM15

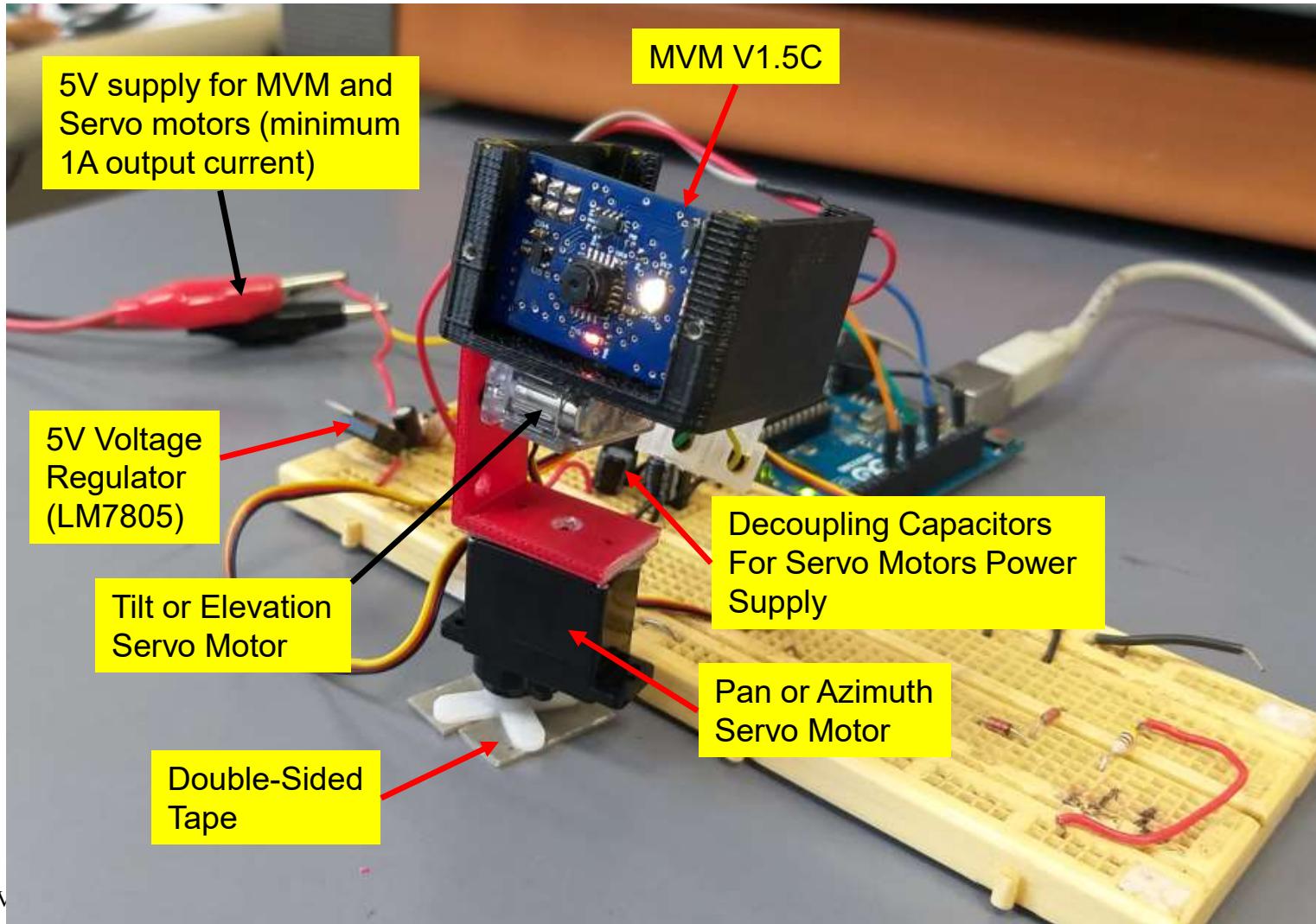
# Example 3 - Color Object Tracking with Arduino Uno

See sample code in GitHub repository

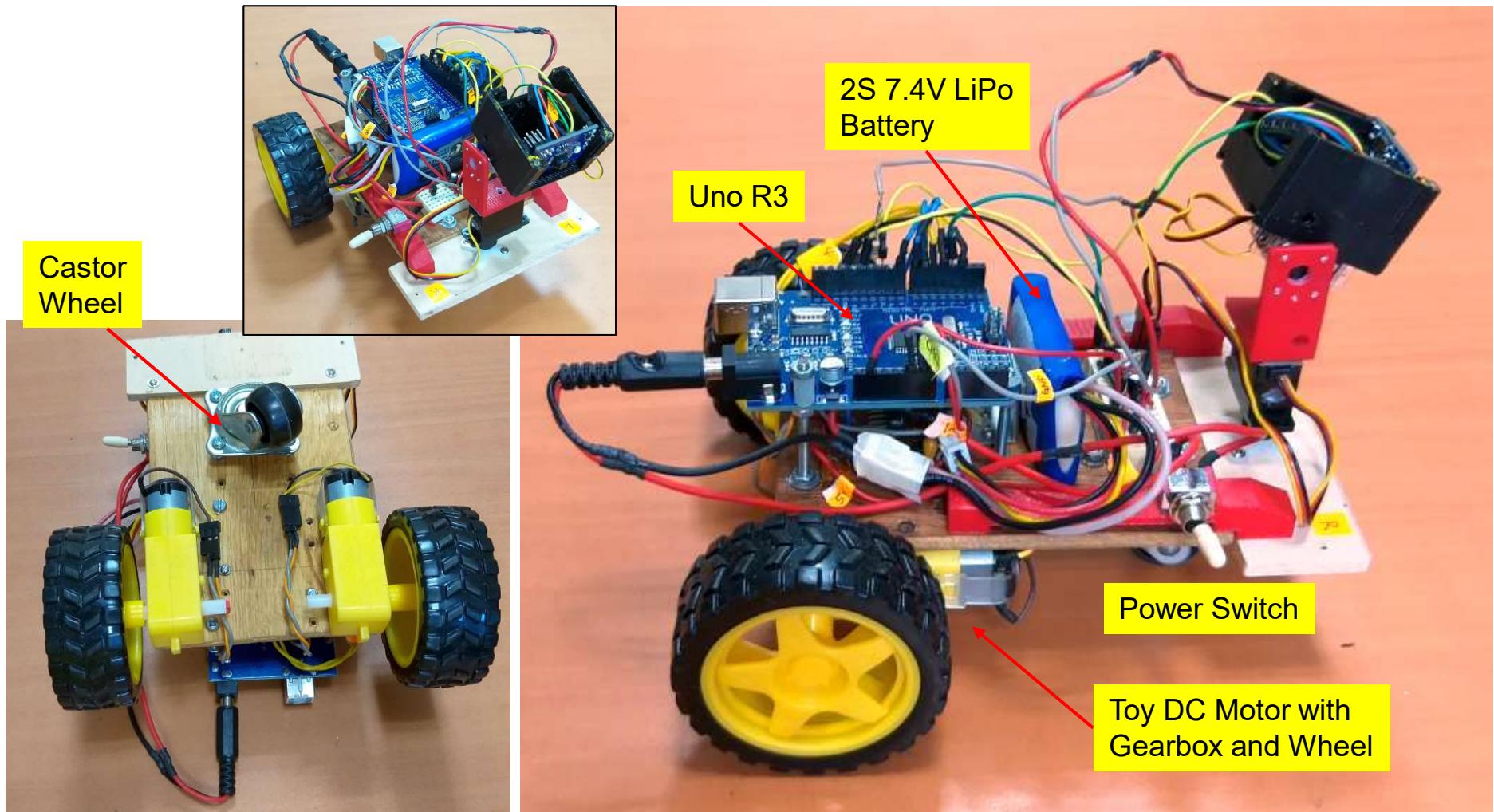
To 5V Power Supply (>1A output)  
or Battery connected to 5V Voltage Regulator



# Example 3 - Mechanical Setup

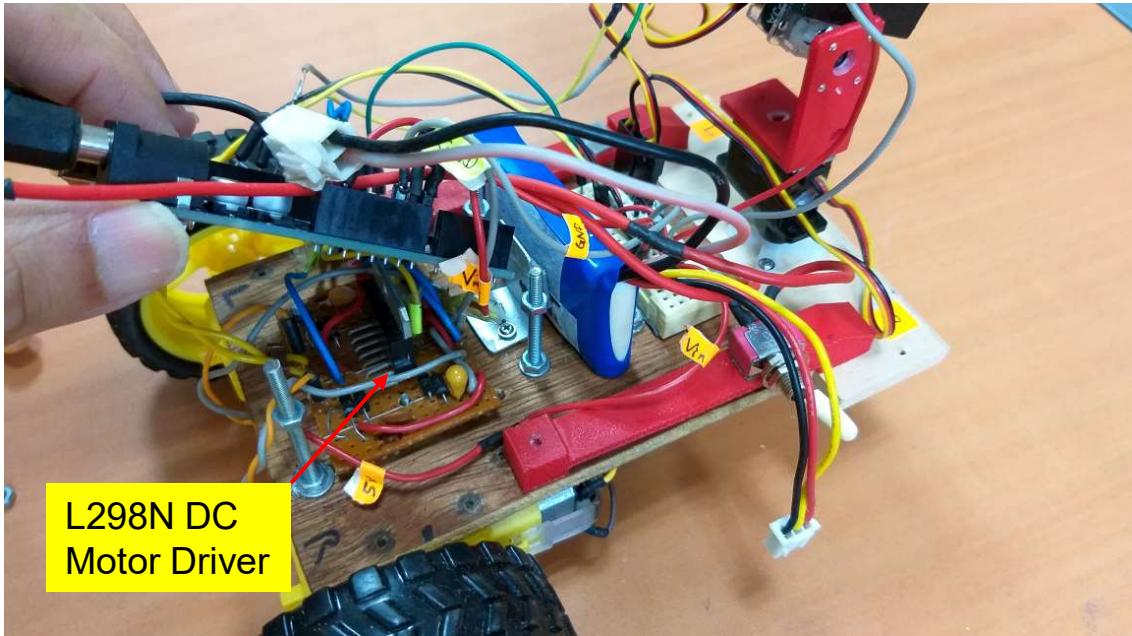


# Example 4 - Autonomous Navigation with Arduino Uno Based Robocar



# Example 4 - Wiring Information of Robocar

---



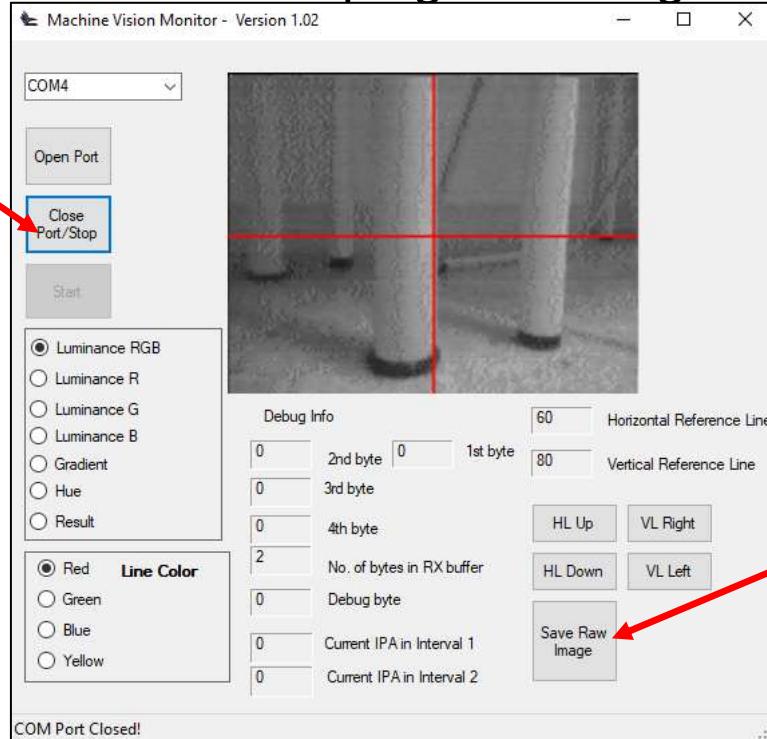
## Arduino Uno R3

- Pin 2, 4 – Left DC motor direction control.
- Pin 7, 8 – Right DC motor direction control.
- Pin 5 – Left DC motor speed control.
- Pin 6 – Right DC motor speed control.
- Pin 10 – RX, to MVM TX pin.
- Pin 11 – TX, to MVM RX pin via  $100\Omega$  resistor.
- Optional: Pin 3 for azimuth servo motor control and Pin 12 for elevation servo motor control.

# Saving the Image Frame onto Computer Harddisk and Retrieving the Image using Scilab or MATLAB software

- As mentioned in slide #13, one can save the image displayed in the Machine Vision Monitor software onto hard disk.
- The image file is saved as a binary file containing 2D array of luminance pixels.
- Scilab or MATLAB software to read the file and display the image. This is useful when one is developing a new algorithm.

1. Close the COM port to stop streaming image



2. Save the current bitmap in display onto hard disk

# Continued...

- The Scilab script to read the saved image file is also provided in the MVM\_V1\_5C folder. The script listing is shown below.

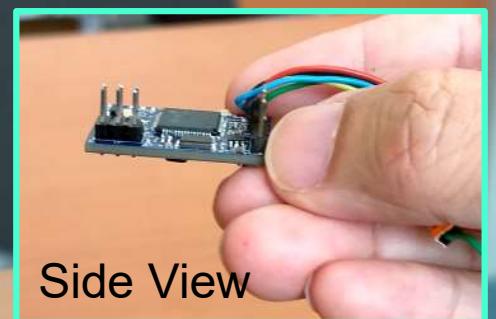
Make sure the path declaration matches your file path in the hard disk

The screenshot shows the Scilab IDE interface. On the left, the code editor displays the script `Basic_GrayScale_GrabImageCamera.sce`. A red arrow points from the text `path = cd("C:\tmp");` to the line where it is assigned the value `"testimage.txt"`. A yellow callout box contains the text: "Make sure the path declaration matches your file path in the hard disk". On the right, a plot window titled "Graphic window number 0" shows a grayscale image of two cylindrical objects, likely test tubes, against a dark background. The x-axis ranges from 0 to 180, and the y-axis ranges from 0 to 140. A large pink arrow points from the code editor towards the plot window.

```
// Author.....: Fabian-Rung
// Last-modified.: 29-Oct-2016
// Purpose.....: Basic code to load a 8-bit grayscale image from file
//
clear;
ImageWidth = 160; // Set the size of the image. QQVGA.
ImageHeight = 120;
Hgraf = scf(); // Get the handle to current graphic window.
path = cd("C:\tmp"); // Path for the image file.
Hfile = mopen("testimage.txt",'rb'); // Open a text file for reading
// (don't skip 0x0D, newline character)
M = zeros(ImageHeight+1,ImageWidth+1); // Matrix to hold the gray scale image data.
Mt = zeros(ImageWidth+1,ImageHeight+1); // Another matrix also to hold the gray scale image data.
// 't' indicate transpose.
for i=1:ImageHeight-1 // ImageHeight-1 due to the last line is not
// exported from the camera-monitor software
    for j=1:ImageWidth
        M(i,j) = mget(1,'c',Hfile); // Read 1 pixel data, convert to double.
    end
    mget(1,'c',Hfile); // Read the newline/carriage return character.
end
// Transpose and flip the image so that
// it appear at the correct orientation.
Mt(j,ImageHeight-i) = M(i,j); // The original format of Mt[] is:
// -----> Column
// -----
// -----
// V
// Row
mclose(Hfile); // Release file handle.
// Hgraf.color_map = graycolormap(127); // Set current graphic window color map.
Hgraf.color_map = graycolormap(255); // to gray scale, 127 or 255 levels.
row = 1:ImageWidth + 1;
col = 1:ImageHeight + 1;
grayplot(row,col,Mt); // Plot the transpose and flip image.
```

---

MVM V1.5C  
installed on  
robot



Side View

