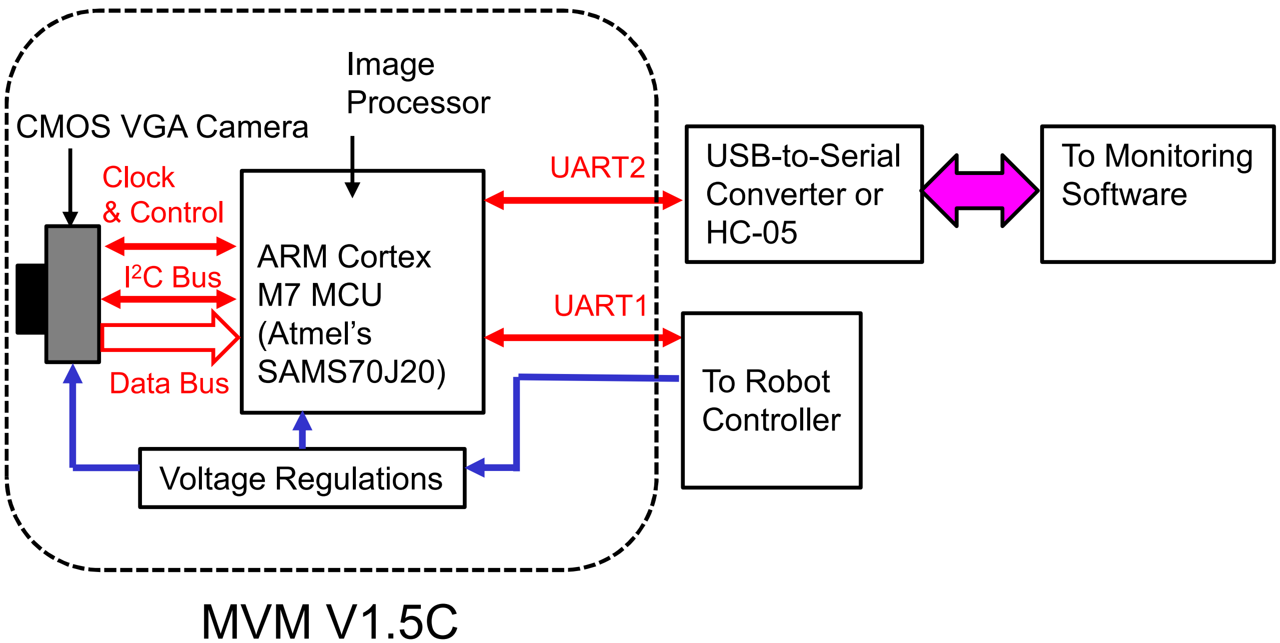
MVM V1.5C Quick Start Guide

Rev 0.95

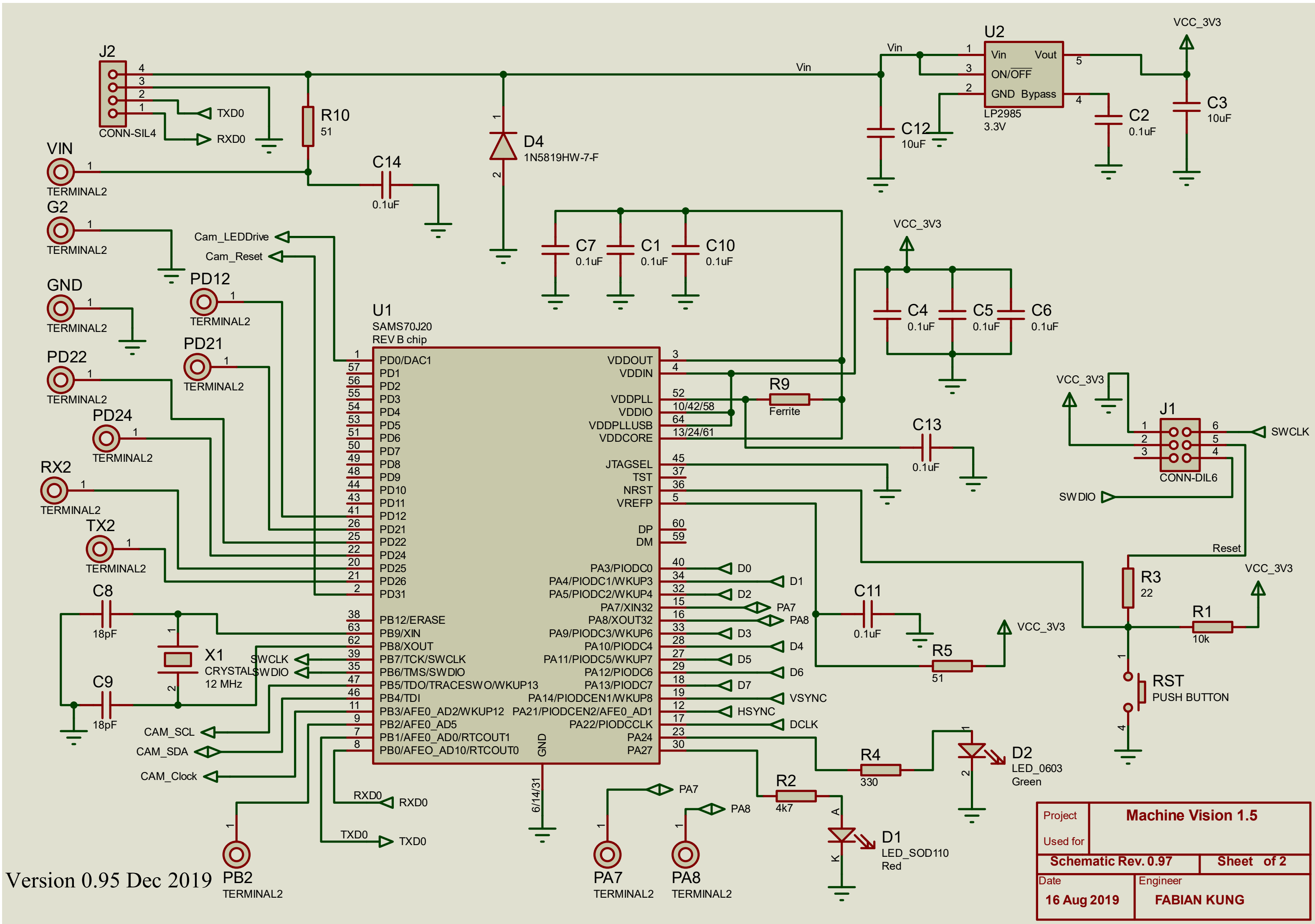
What Is It?

- 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.
- 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!)

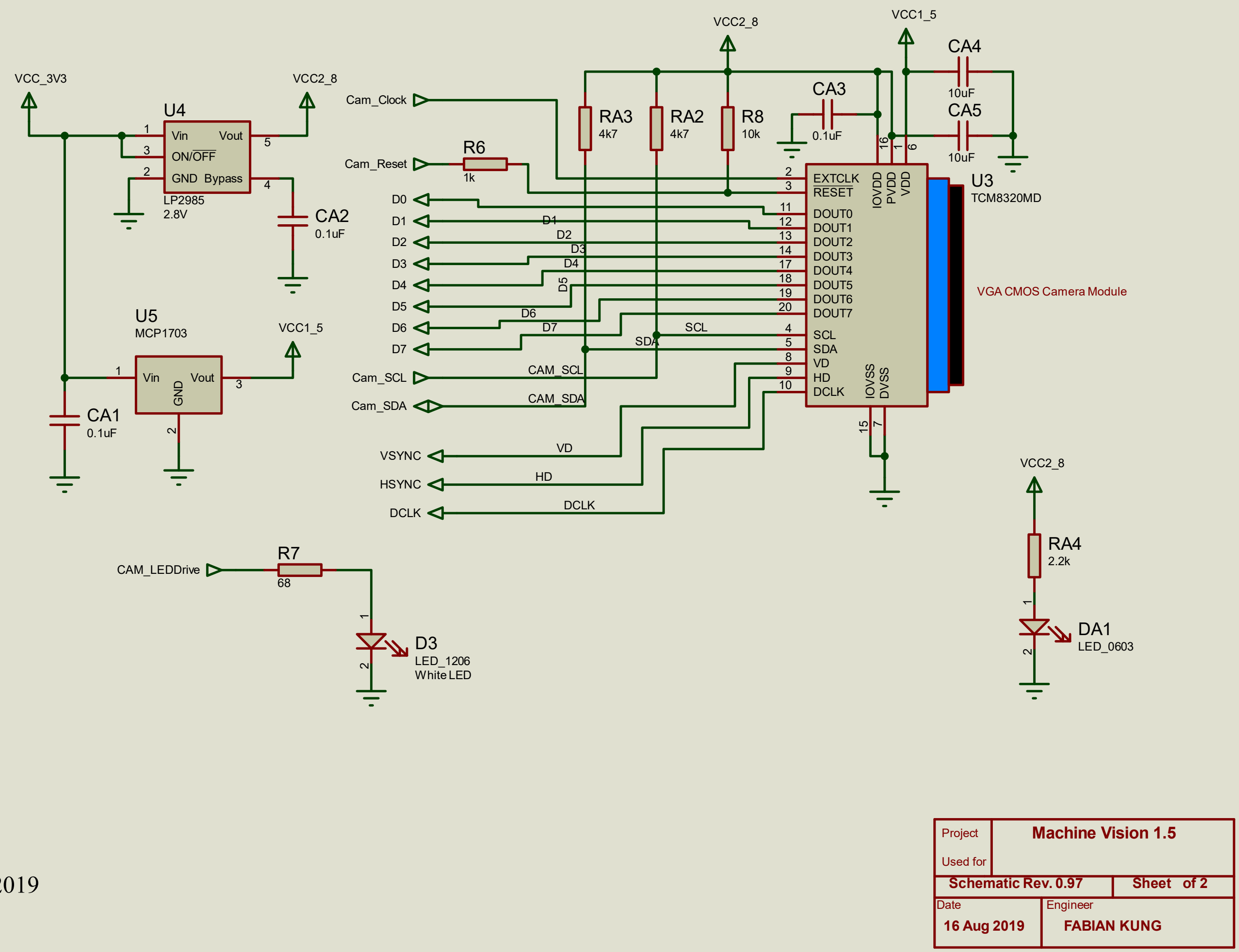
Block Diagram



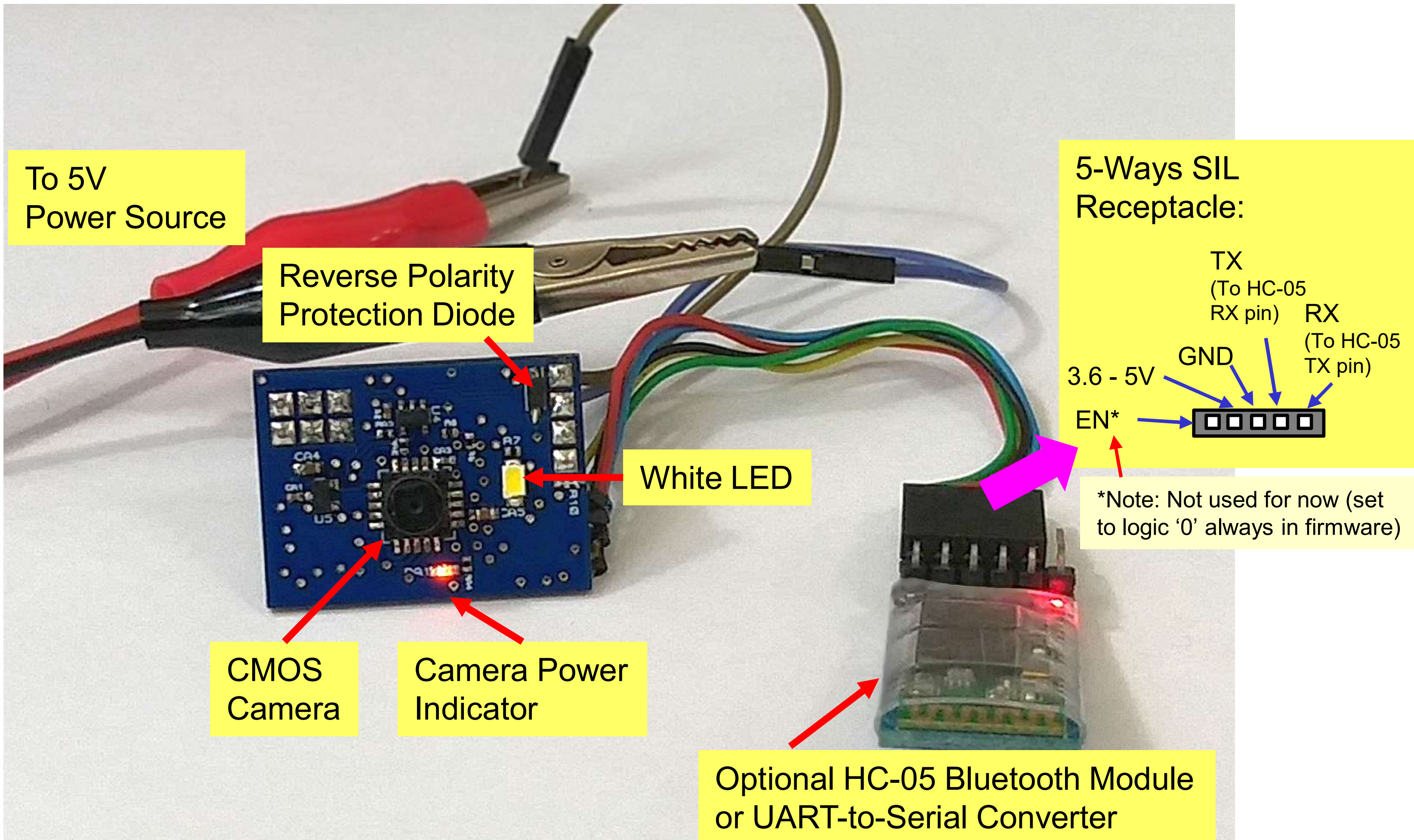
Schematic 1 – Micro-controller Core



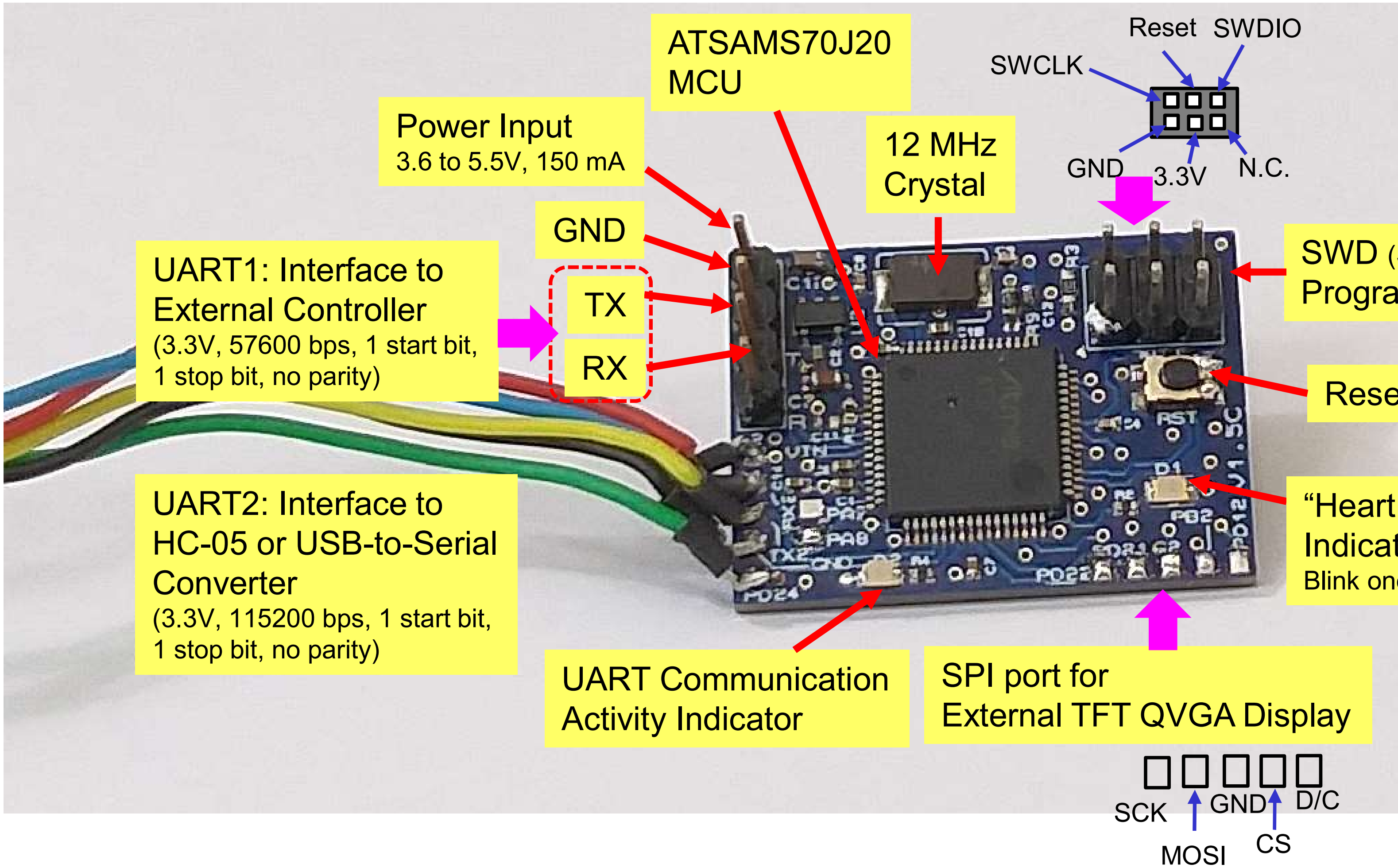
Schematic 2 – Camera Sub-Circuit



Rear View (MVM V1.5C)



Front View (MVM V1.5C)



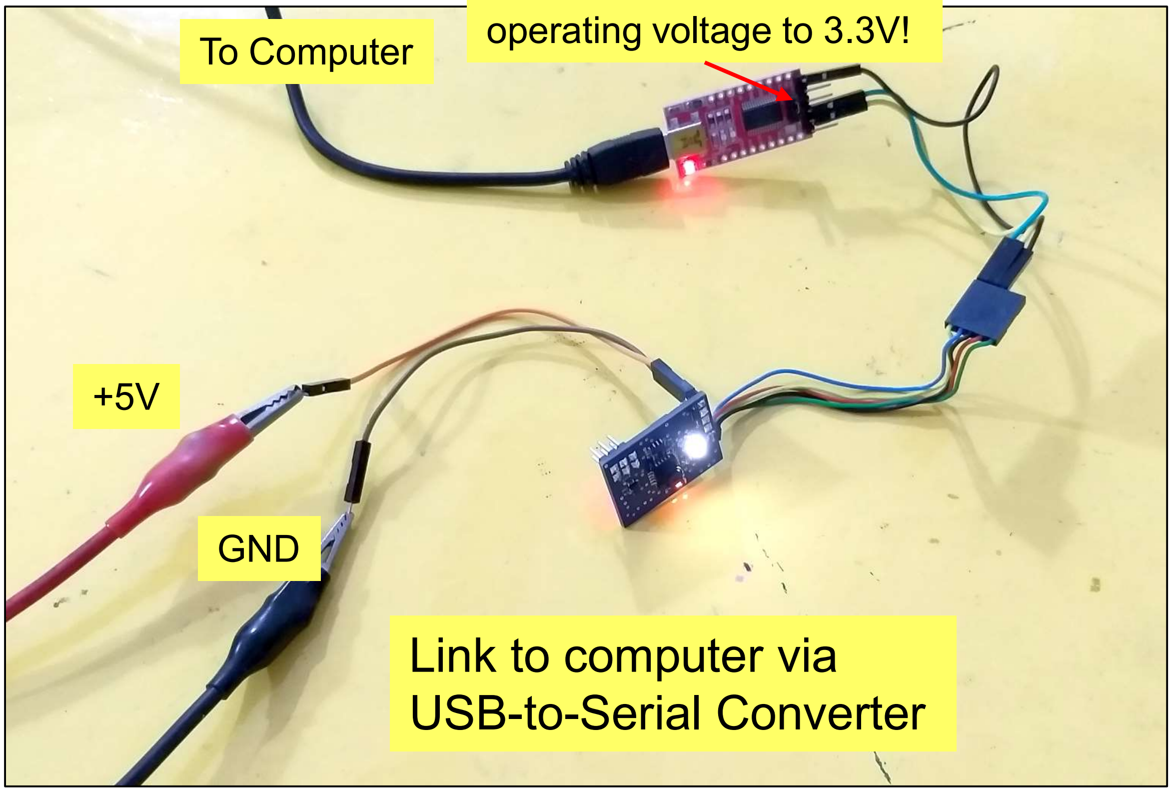
Files

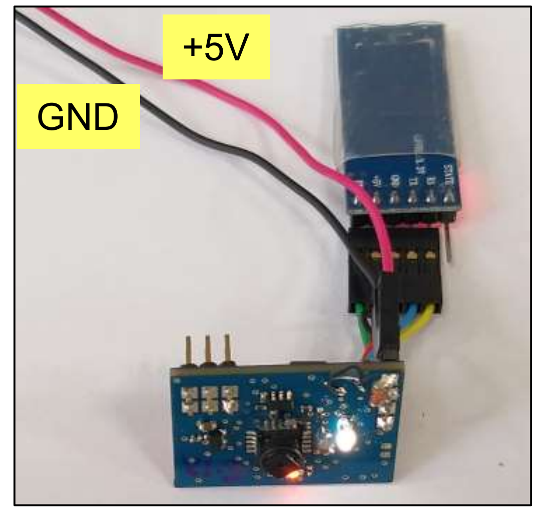
- All relevant files can be obtained from https://github.com/fabiankung/MVM_V1_5C
- Firmware is build using **Atmel Studio 7**.
- PC software is build using **Visual Studio Community 2017** or later.

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.



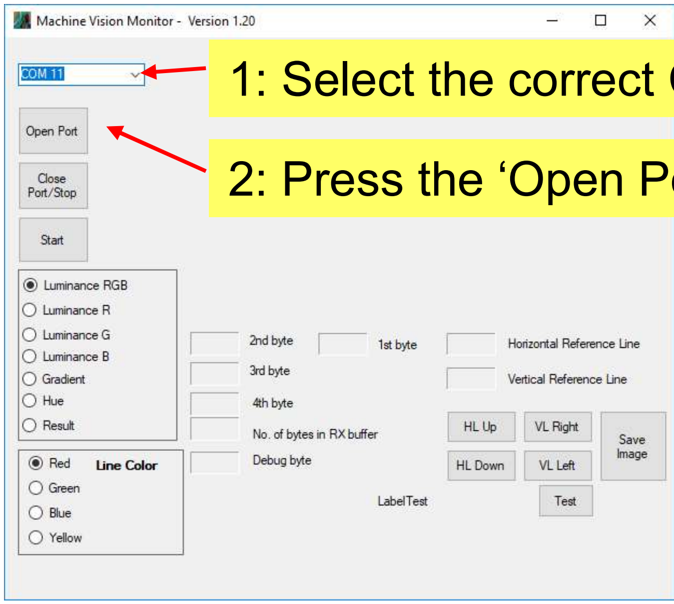


Link to computer via
HC-05 Bluetooth Module

Step 2 – Pair Computer to HC-05

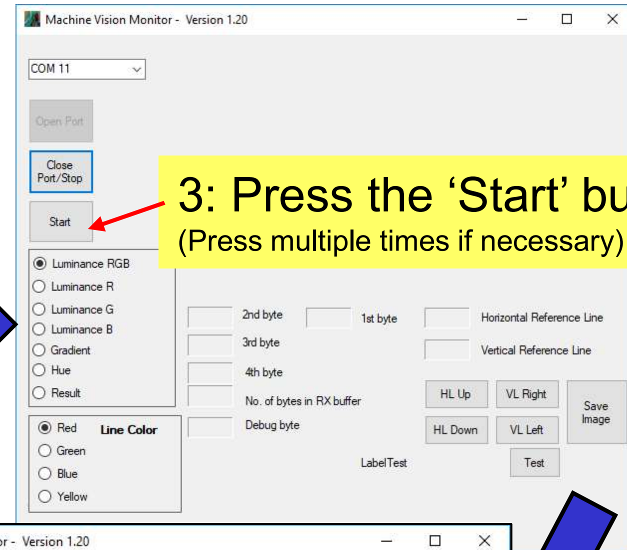
- If need to pair the computer to HC-05.
- Then check virtual COM port number on the computer (for instance by going to the Device Manager).

Step 3 – Run the Machine Vision Monitor Software (MV_Monitor.exe)

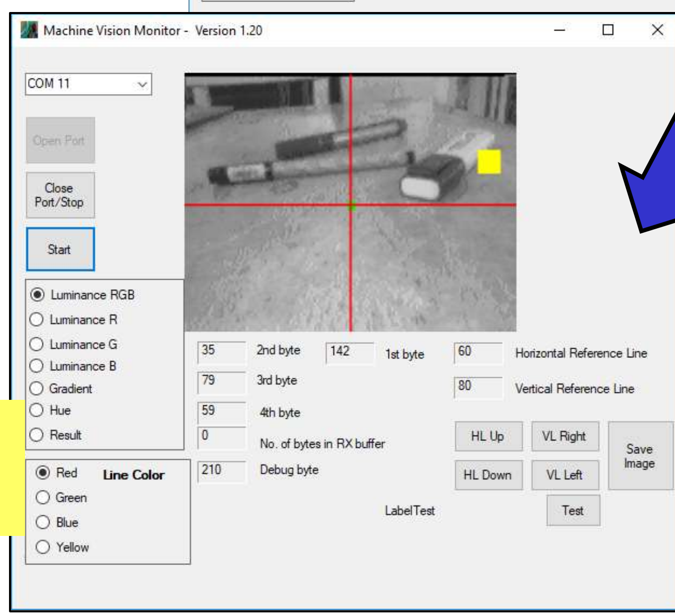


1: Select the correct COM port

2: Press the 'Open Port' button



3: Press the 'Start' button
(Press multiple times if necessary)



4: Now the MVM will transmit the image data line-by-line to the monitor software

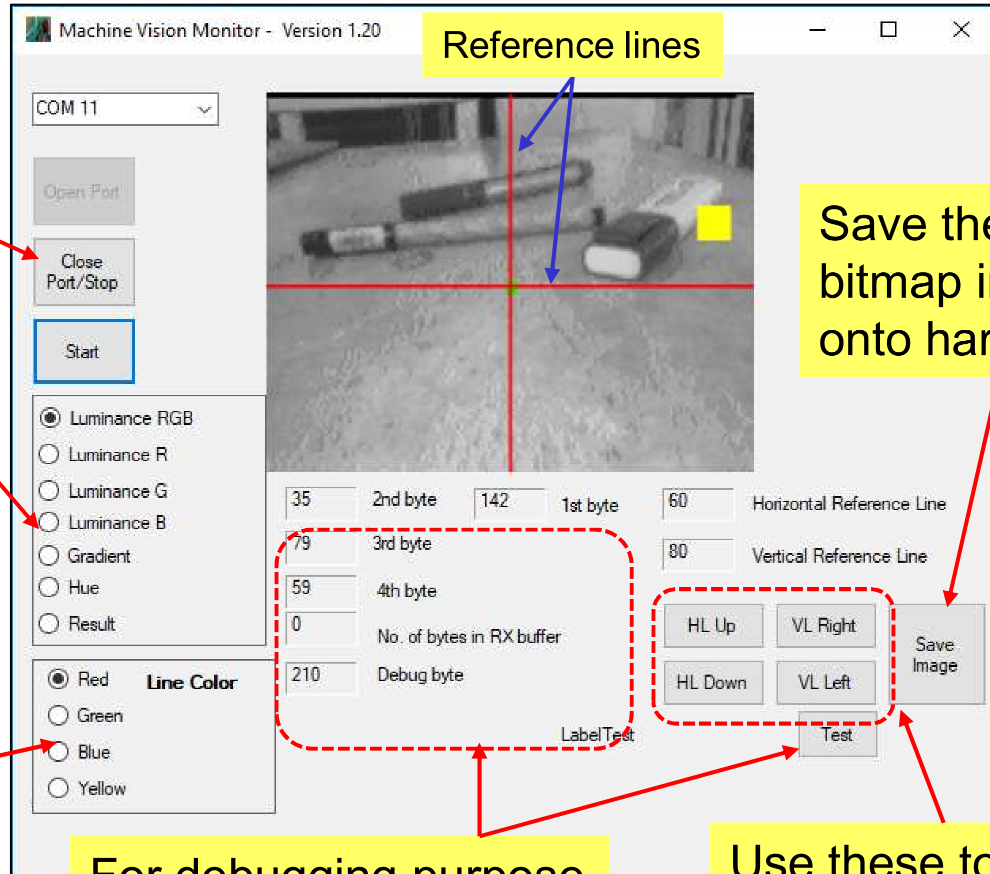
Other Information (1 of 2)

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.

Reference lines

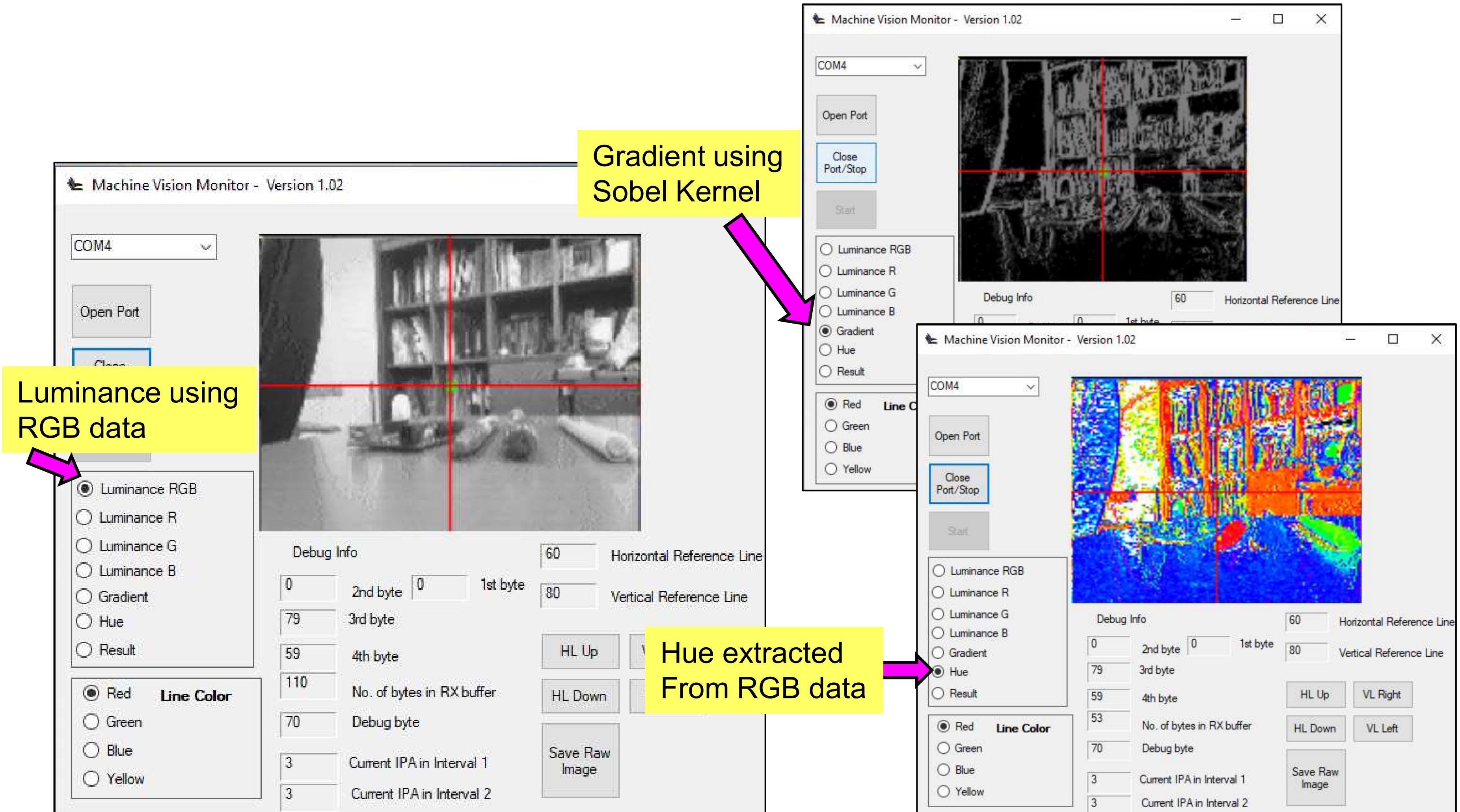


Save the current bitmap in display onto hard disk

For debugging purpose (see source codes)

Use these to move the reference lines.

Other Information (2 of 2)

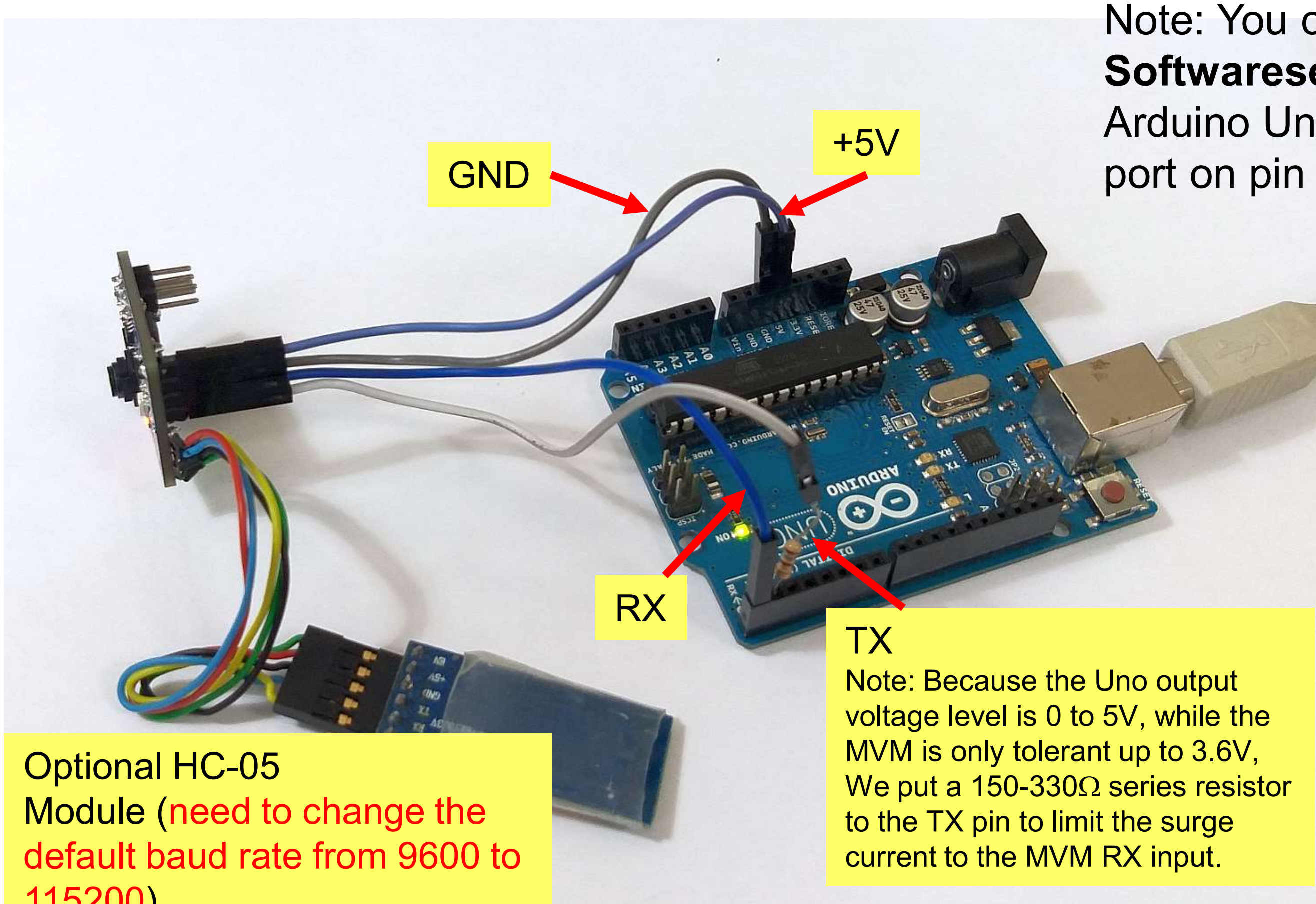


Connection to External Controller for Robotic Projects

Connection to External Controllers

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

Note: You can also use **Softwareserial** instead of Arduino Uno hardware serial port on pin 0 and 1.



TX
Note: Because the Uno output voltage level is 0 to 5V, while the MVM is only tolerant up to 3.6V, We put a 150-330Ω series resistor to the TX pin to limit the surge current to the MVM RX input.

Optional HC-05 Module (need to change the default baud rate from 9600 to 115200)

UART1 Communication Protocol

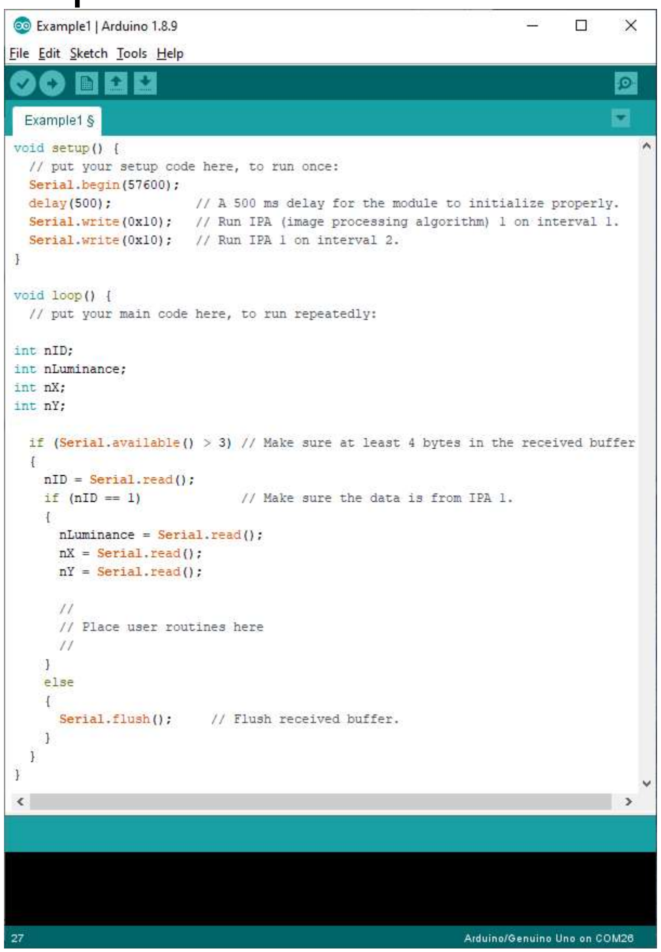
Image Processing Algorithm (IPA)	To Activate	MVM Output
Search for brightest spot in a scene. Image resolution = 160x120	Send hex values to MVM: 0x10 to search for brightest spot	4 bytes: Byte 1 = 1 (Algorithm ID) Byte 2 = Maximum luminance value (1 to 127). Byte 3 = x coordinate of region Byte 4 = y 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 = $0b00000b_2b_1b_0$ Byte 3 = $0b00000b_2b_1b_0$ Byte 4 = $0b00000b_2b_1b_0$
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

255 will be send for (x,y) if region with matching color is not found

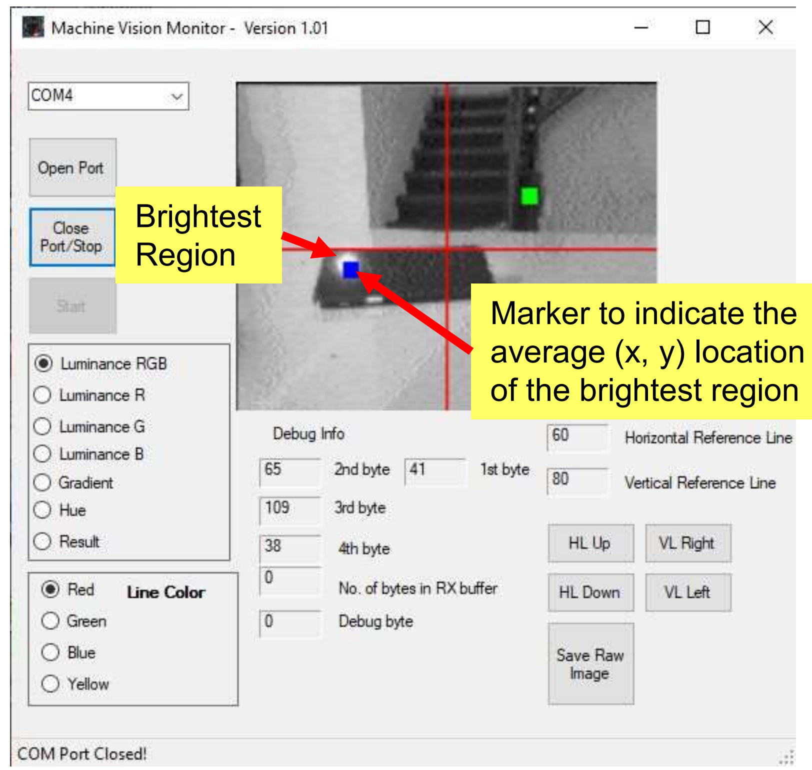
Example 1 – Activate Search for Brightest Spot Algorithm

- 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

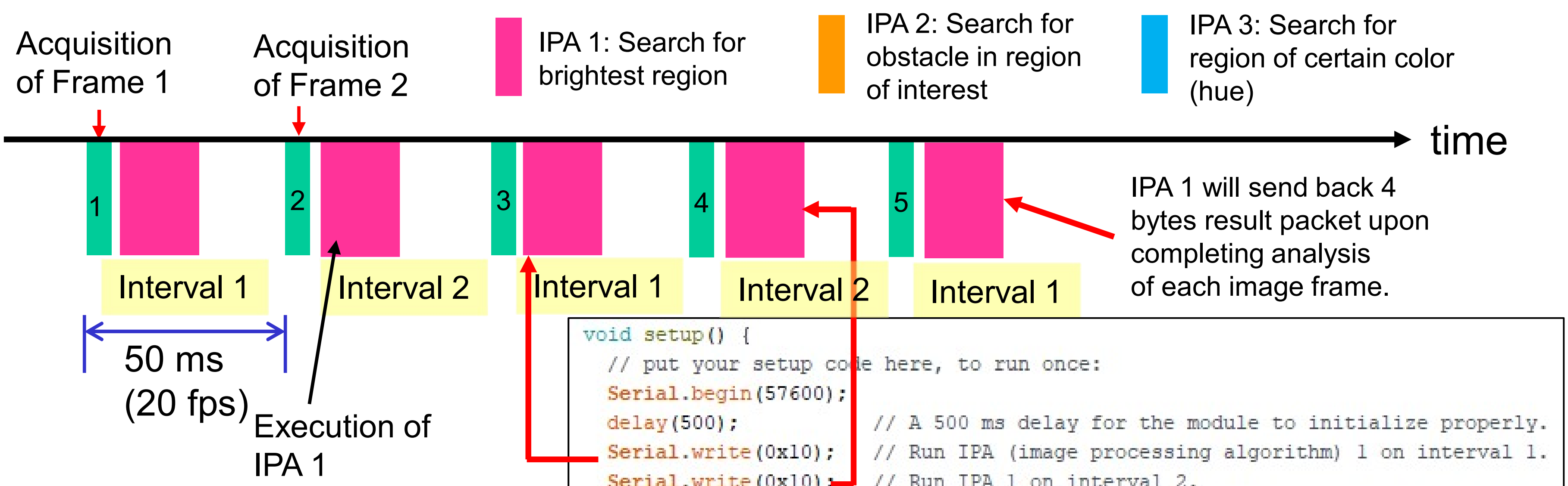



```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(57600);
  delay(500);           // A 500 ms delay for the module to initialize properly.
  Serial.write(0x10);   // Run IPA (image processing algorithm) 1 on interval 1.
  Serial.write(0x10);   // Run IPA 1 on interval 2.
}

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.
    }
  }
}
```



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.

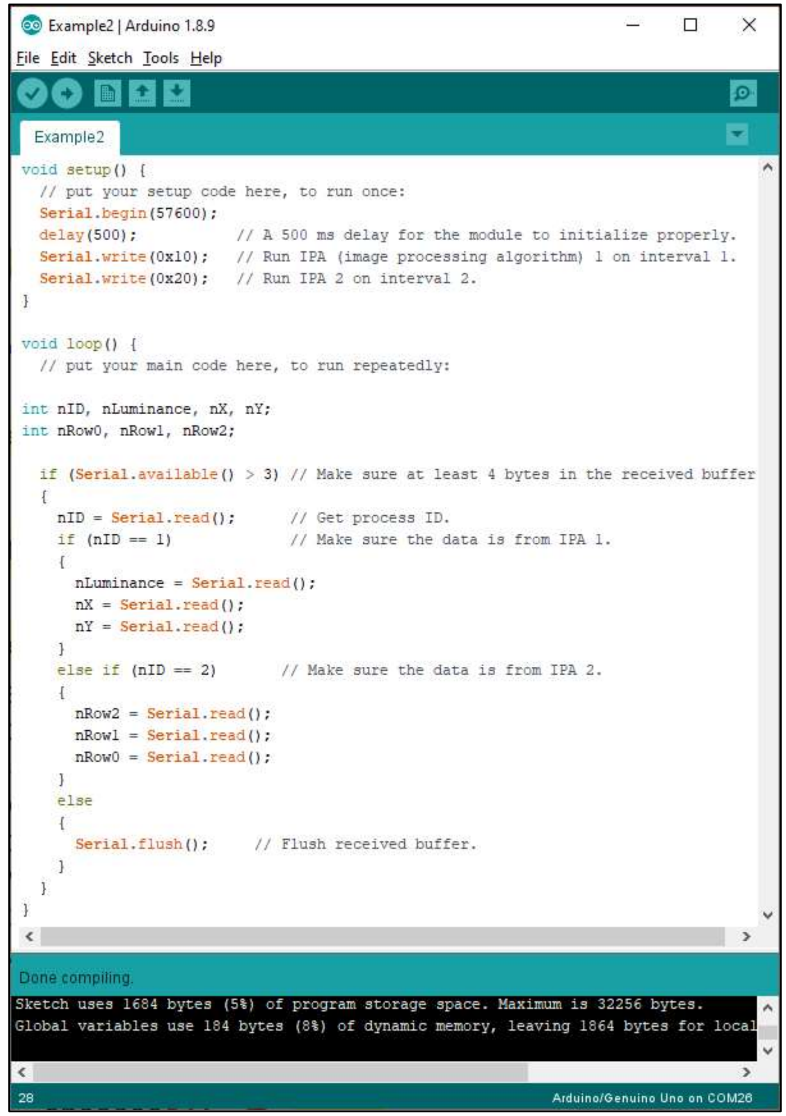



```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(57600);
  delay(500);           // A 500 ms delay for the module to initialize properly.
  Serial.write(0x10);   // Run IPA (image processing algorithm) 1 on interval 1.
  Serial.write(0x10);   // Run IPA 1 on interval 2.
}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

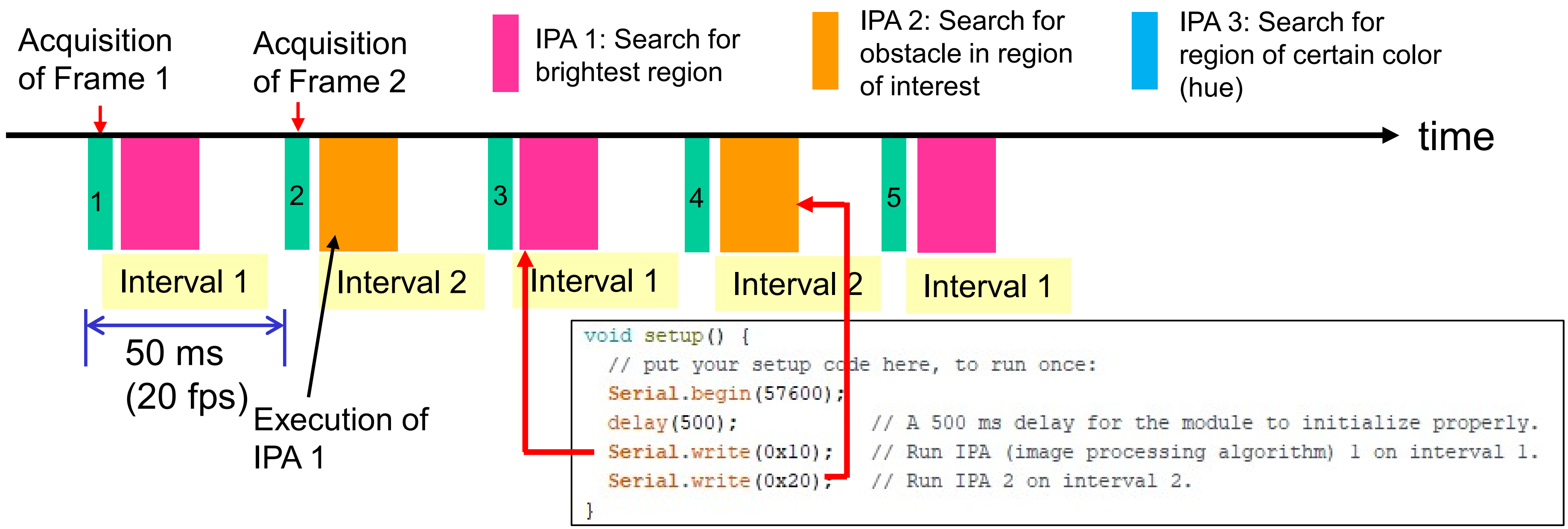
- The C code snippet attaches IPA 1 to both Interval 1 and Interval 2 of the execution flow, thus in this setting IPA 1 runs every 50 ms and any changes in scene is detected within 50 ms.

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.



Example 2 – The Assignment of IPAs to Intervals

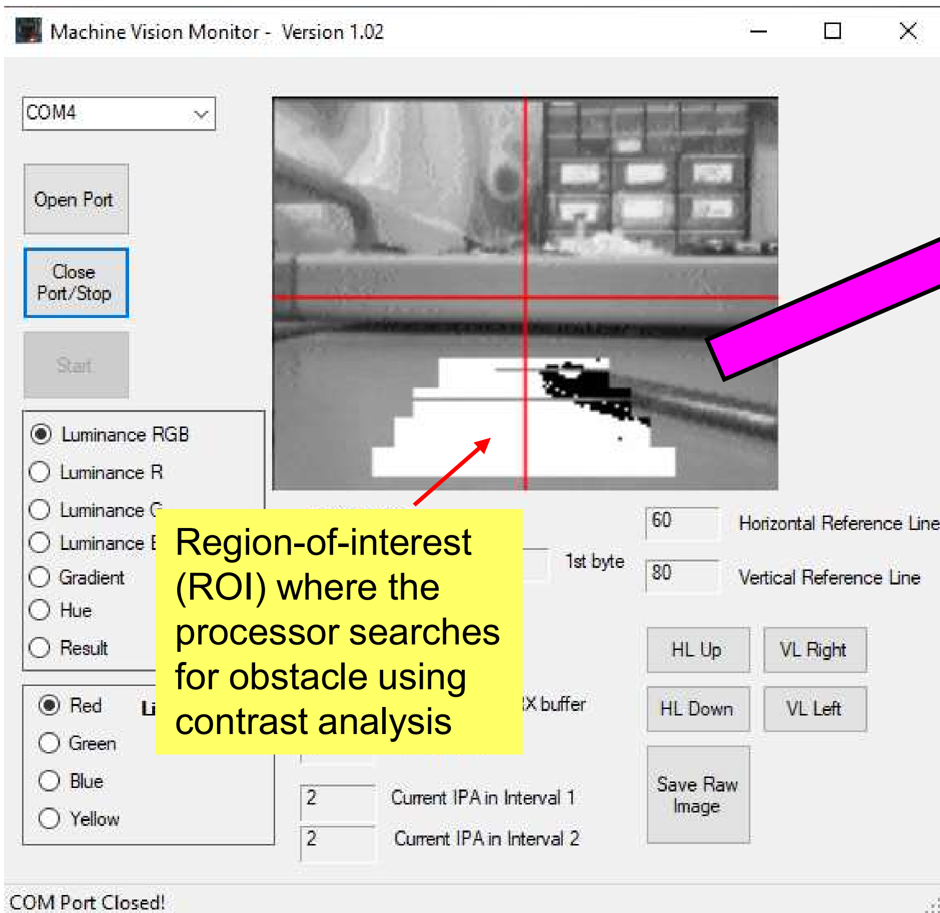


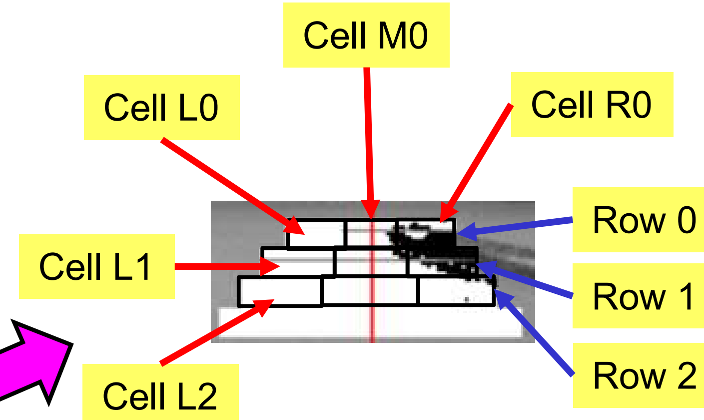

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(57600);
  delay(500);          // A 500 ms delay for the module to initialize properly.
  Serial.write(0x10);  // Run IPA (image processing algorithm) 1 on interval 1.
  Serial.write(0x20);  // Run IPA 2 on interval 2.
}
```

- 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:





Row 3 is not used for now.

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

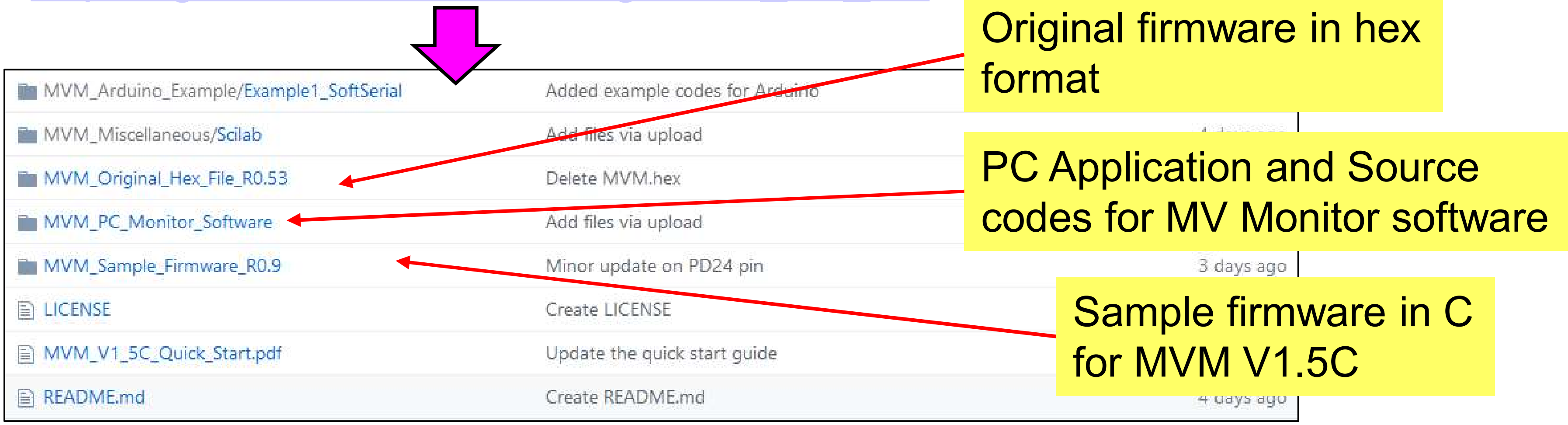
Compiling and Building Your Own Firmware for MVM V1.5C

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

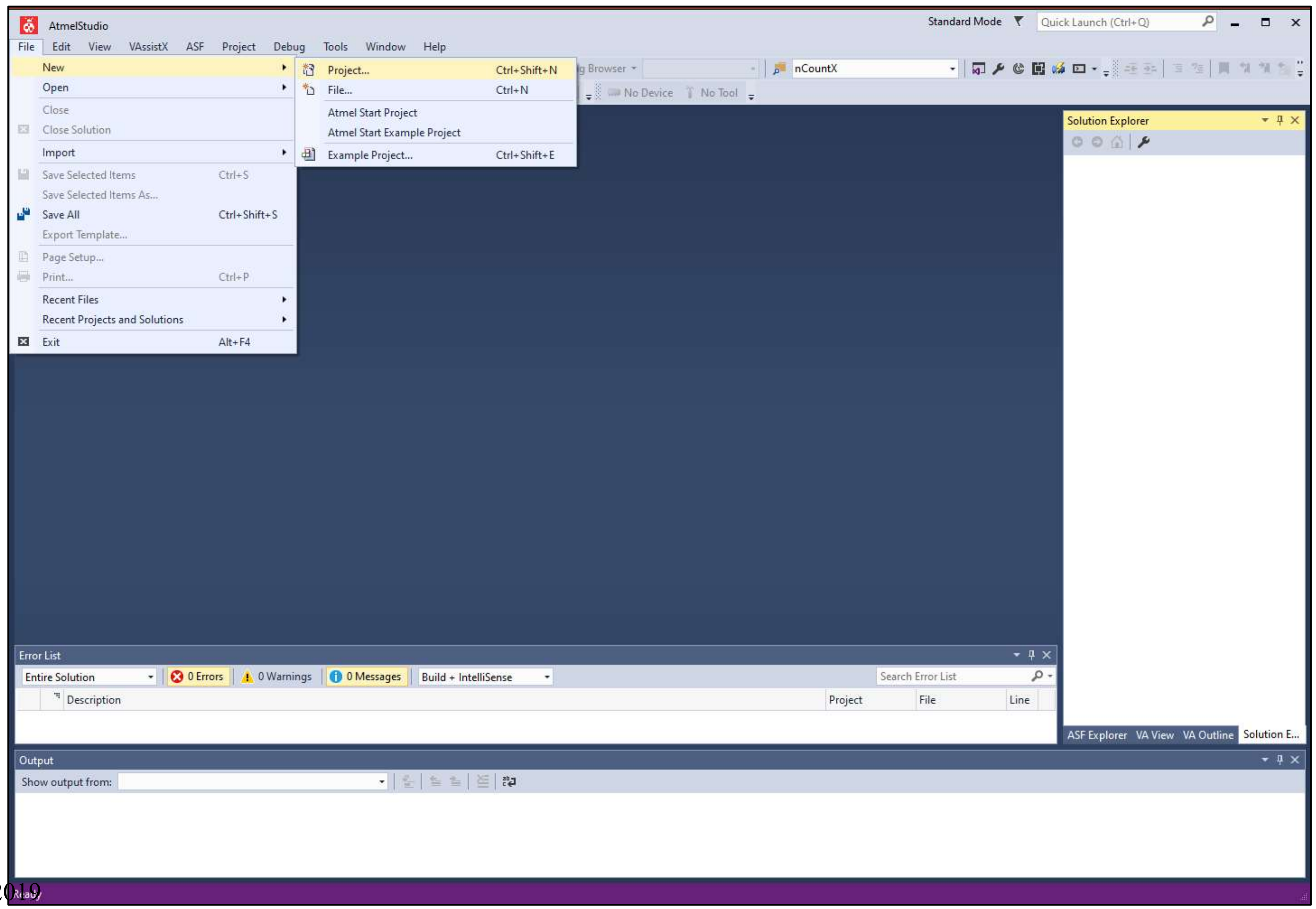
- Clone the MVM_V1_5C folder from https://github.com/fabiankung/MVM_V1_5C



- “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.

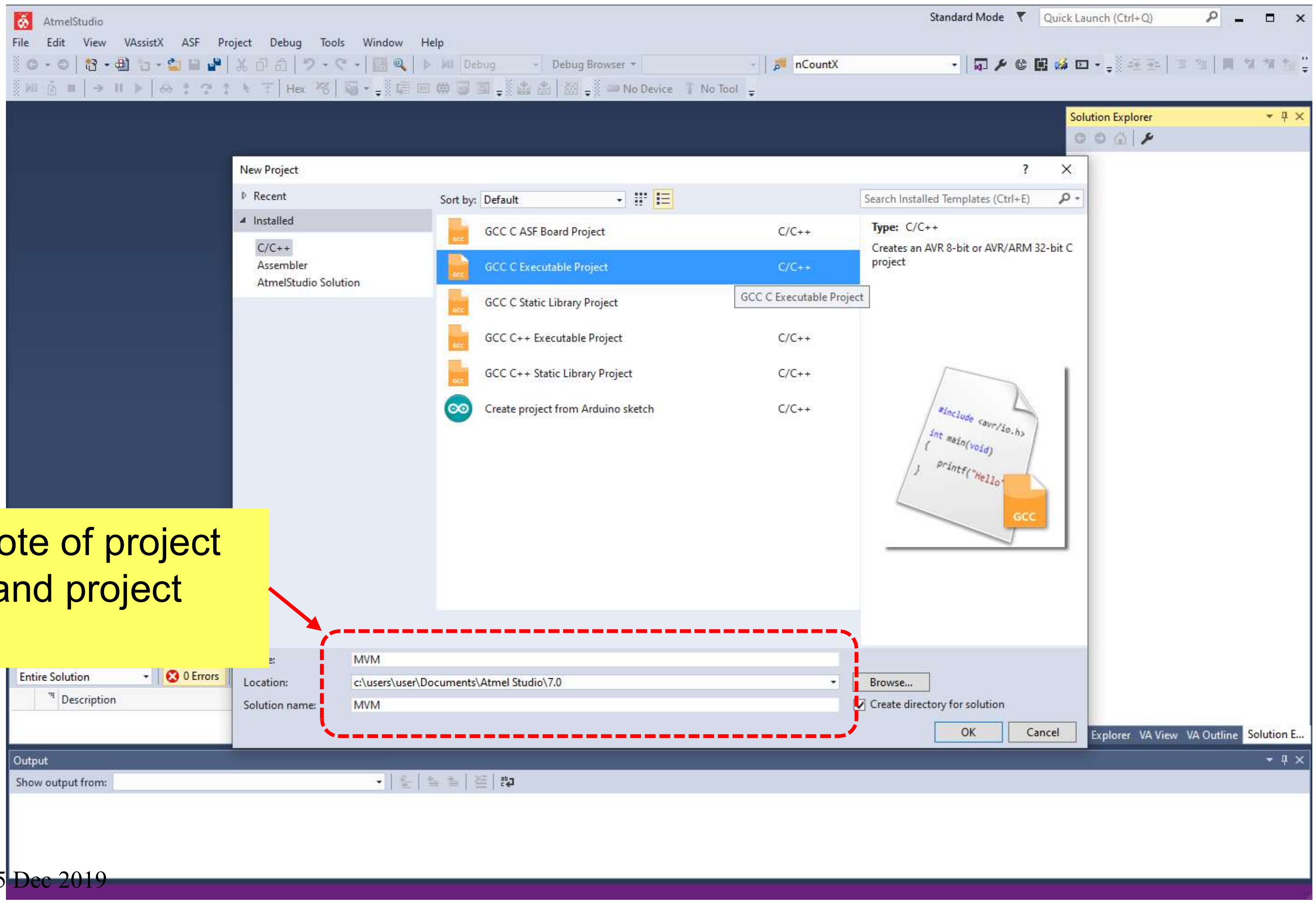
Setting Up An Atmel Studio Project 1

- Start a new project in Atmel Studio 7.



Setting Up An Atmel Studio Project 2

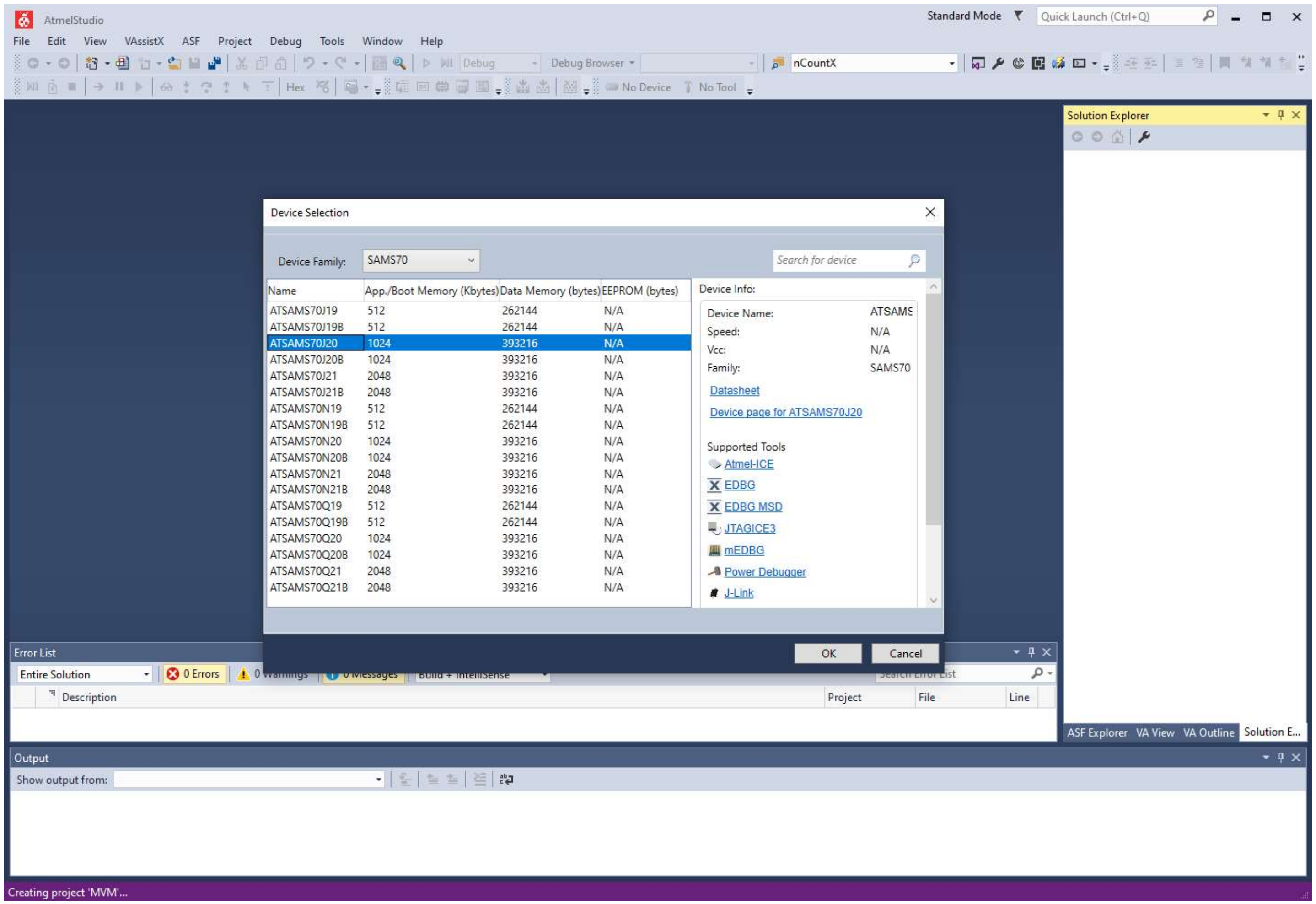
- Create a GCC C executable project.



Take note of project name and project folder

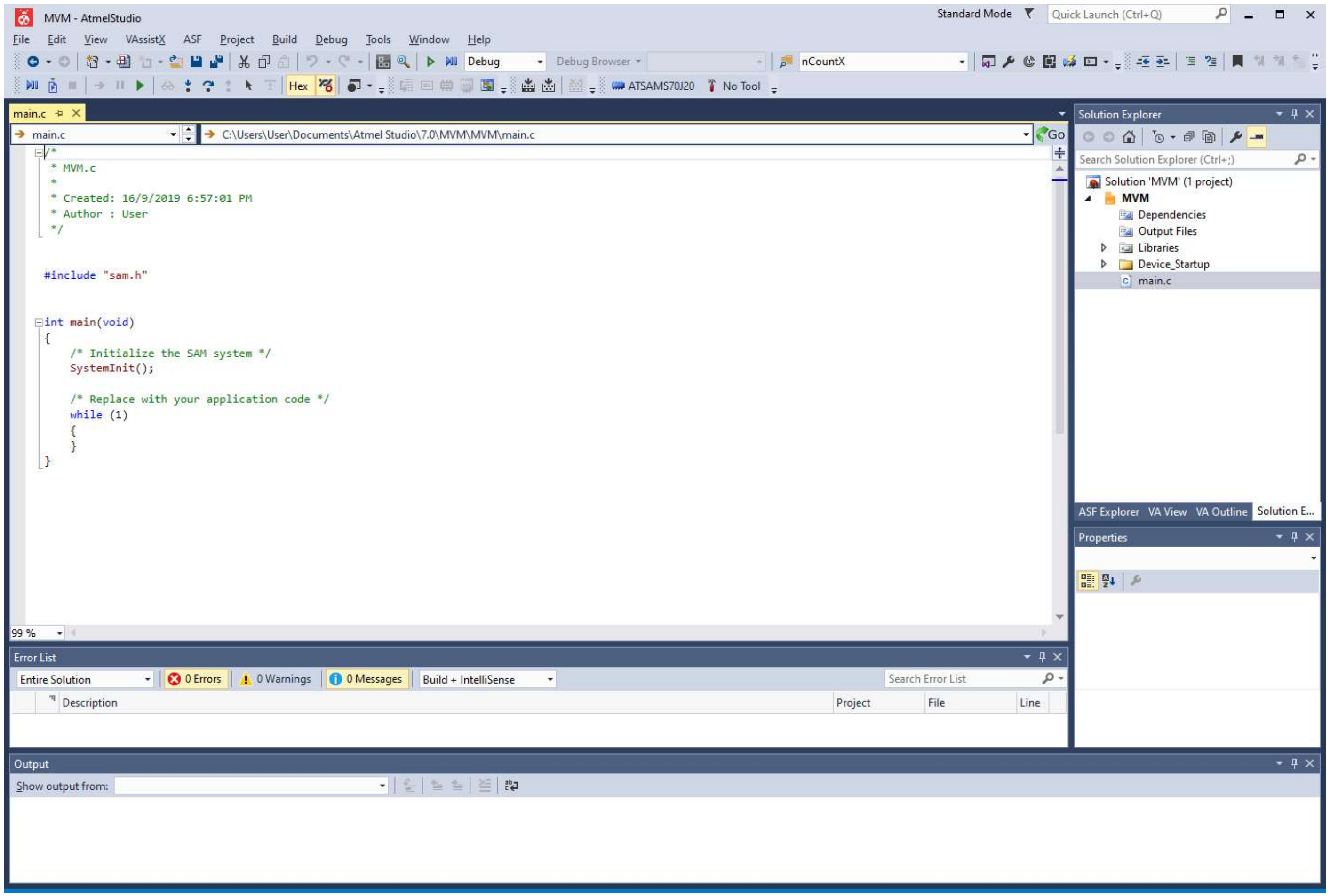
Setting Up An Atmel Studio Project 3

- Select the correct device.



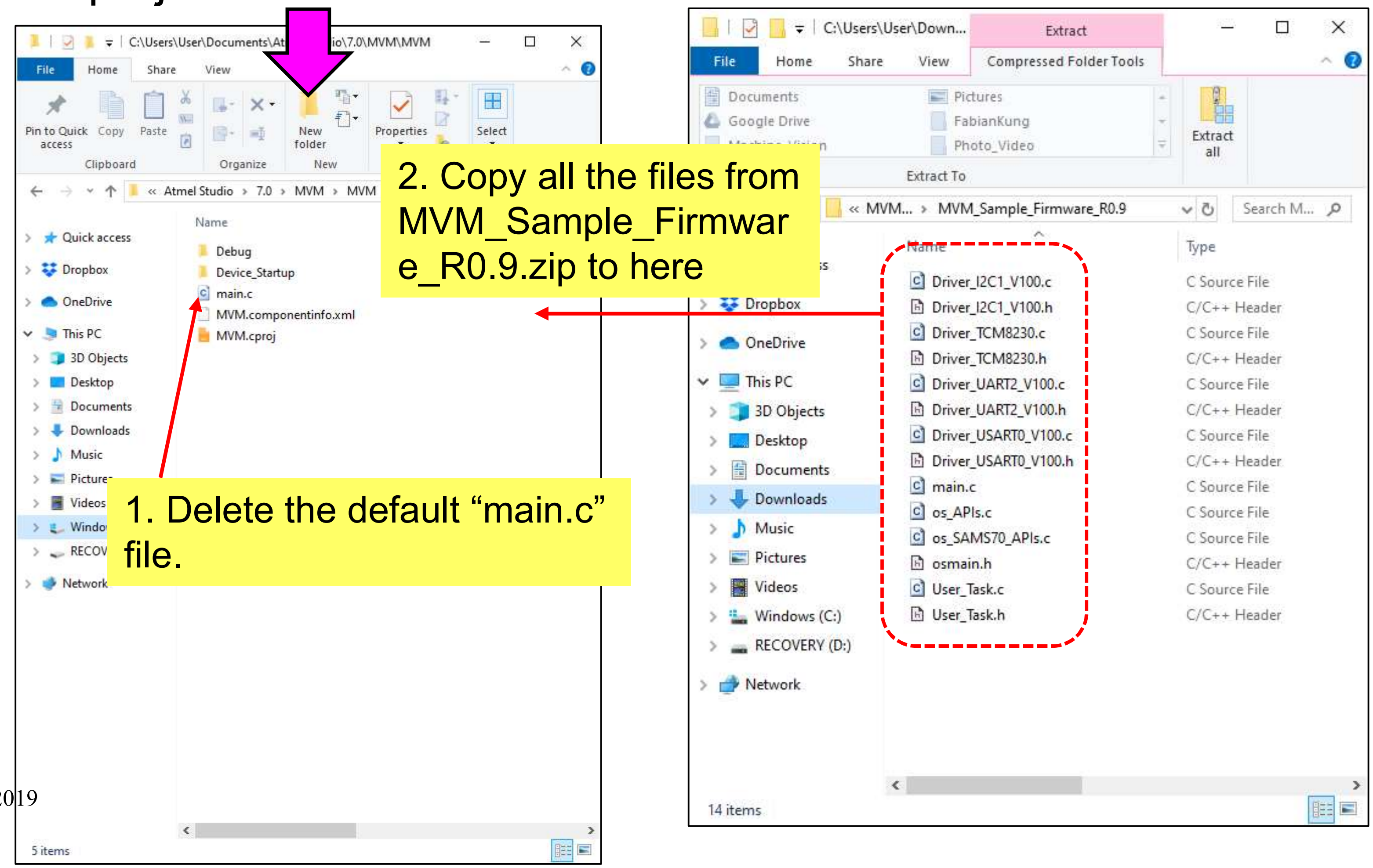
Setting Up An Atmel Studio Project 4

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



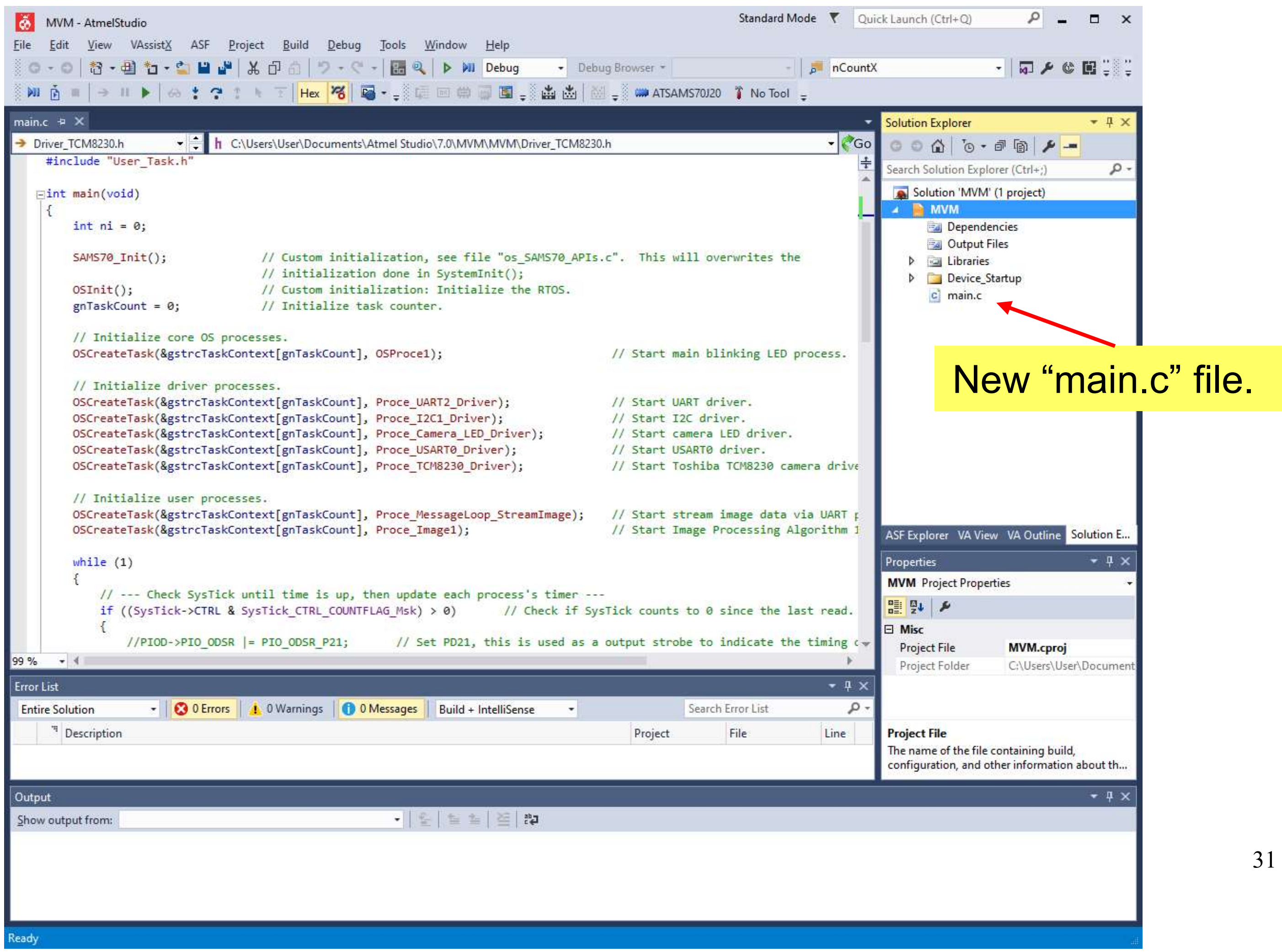
Setting Up An Atmel Studio Project 5

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



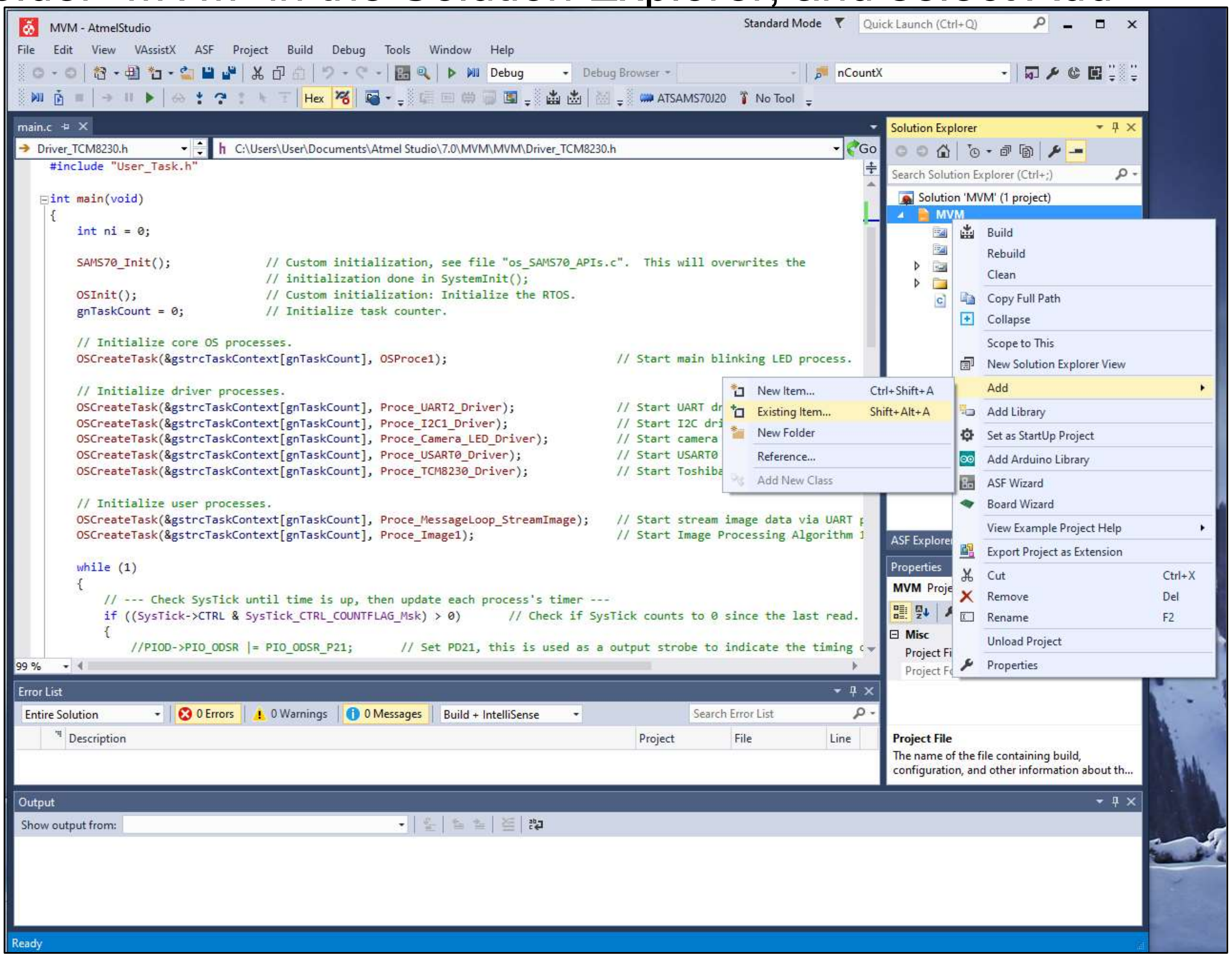
Setting Up An Atmel Studio Project 6

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



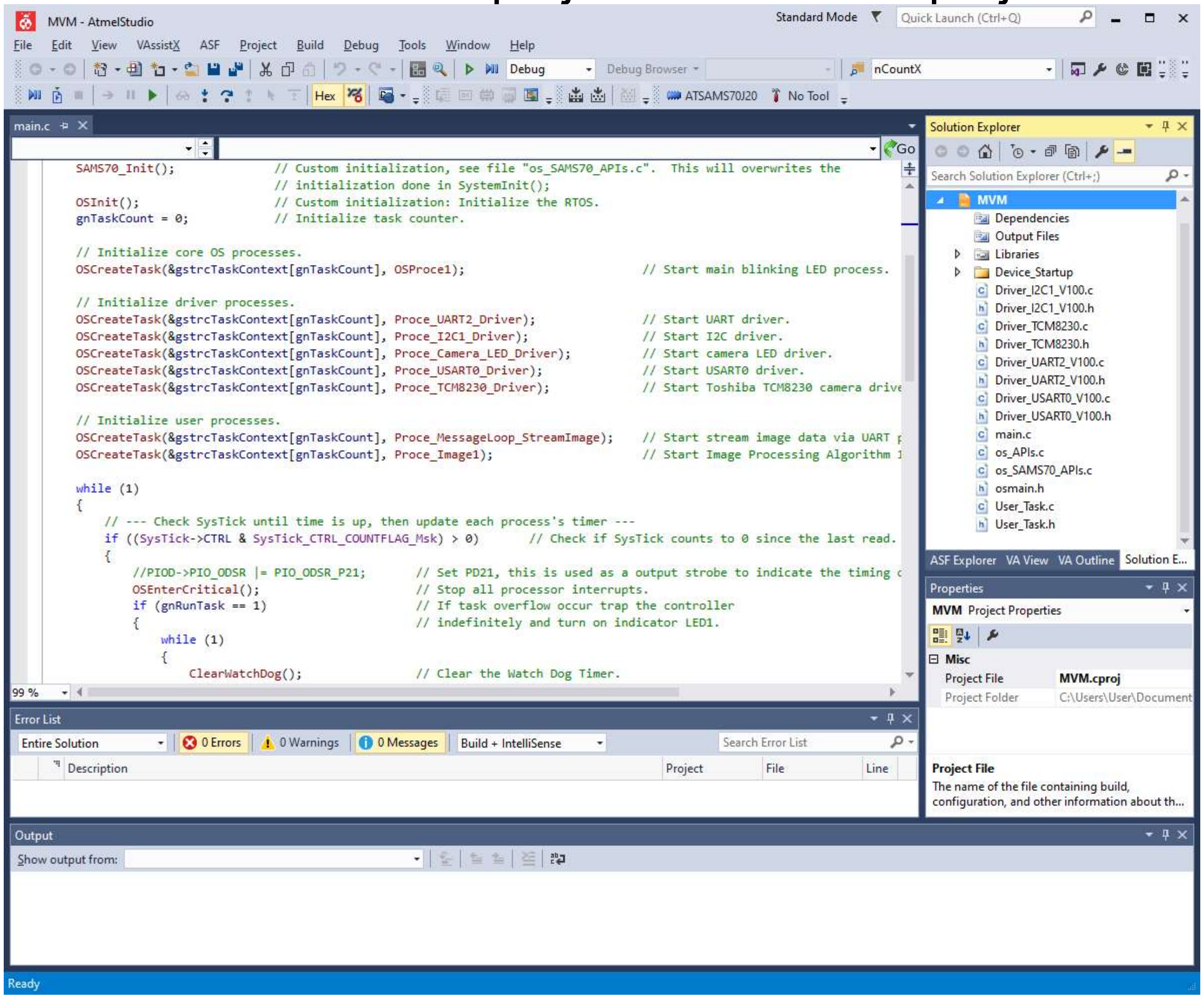
Setting Up An Atmel Studio Project 7

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



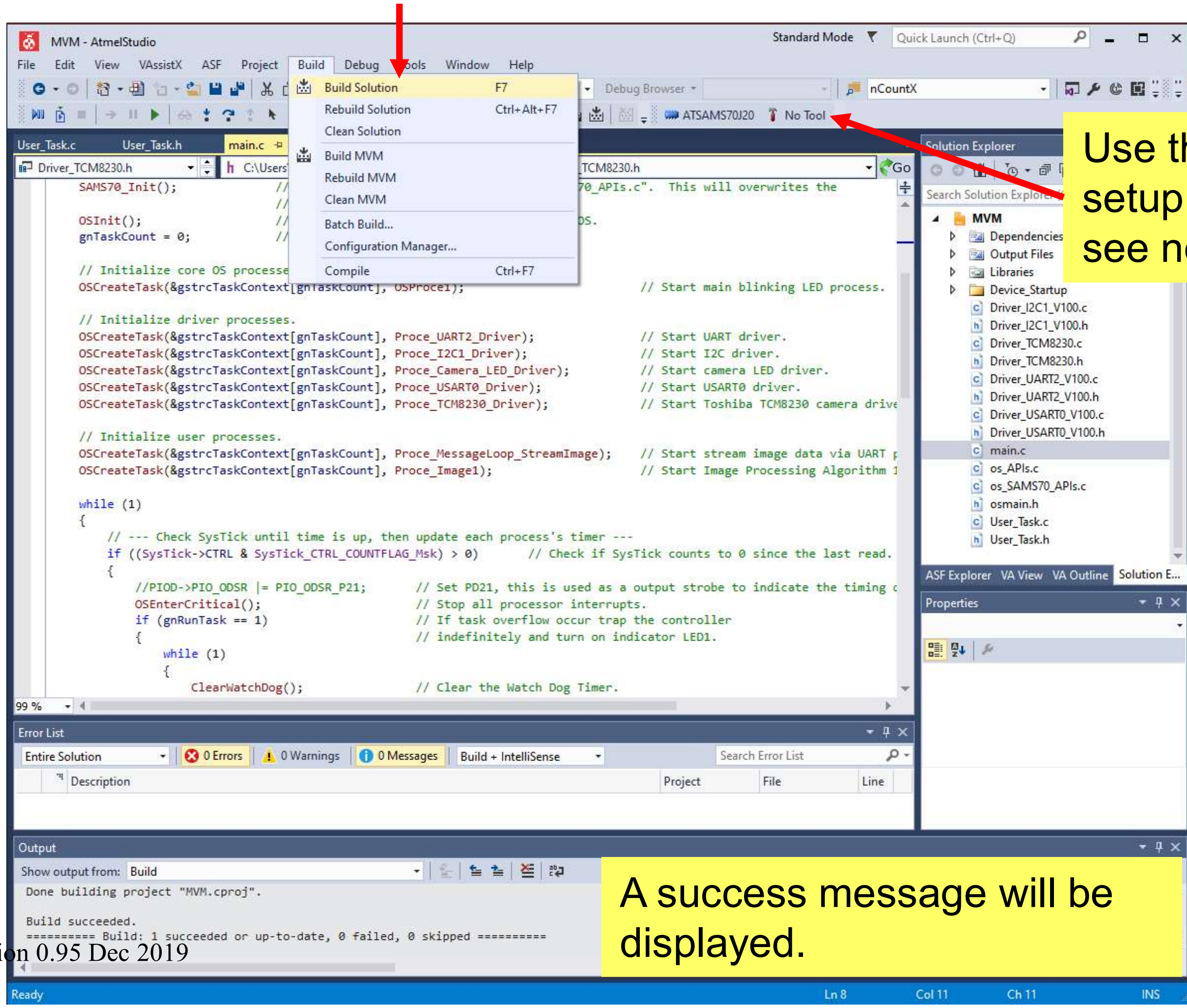
Setting Up An Atmel Studio Project 8

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



Setting Up An Atmel Studio Project 9

- Now you can build or compile the project.

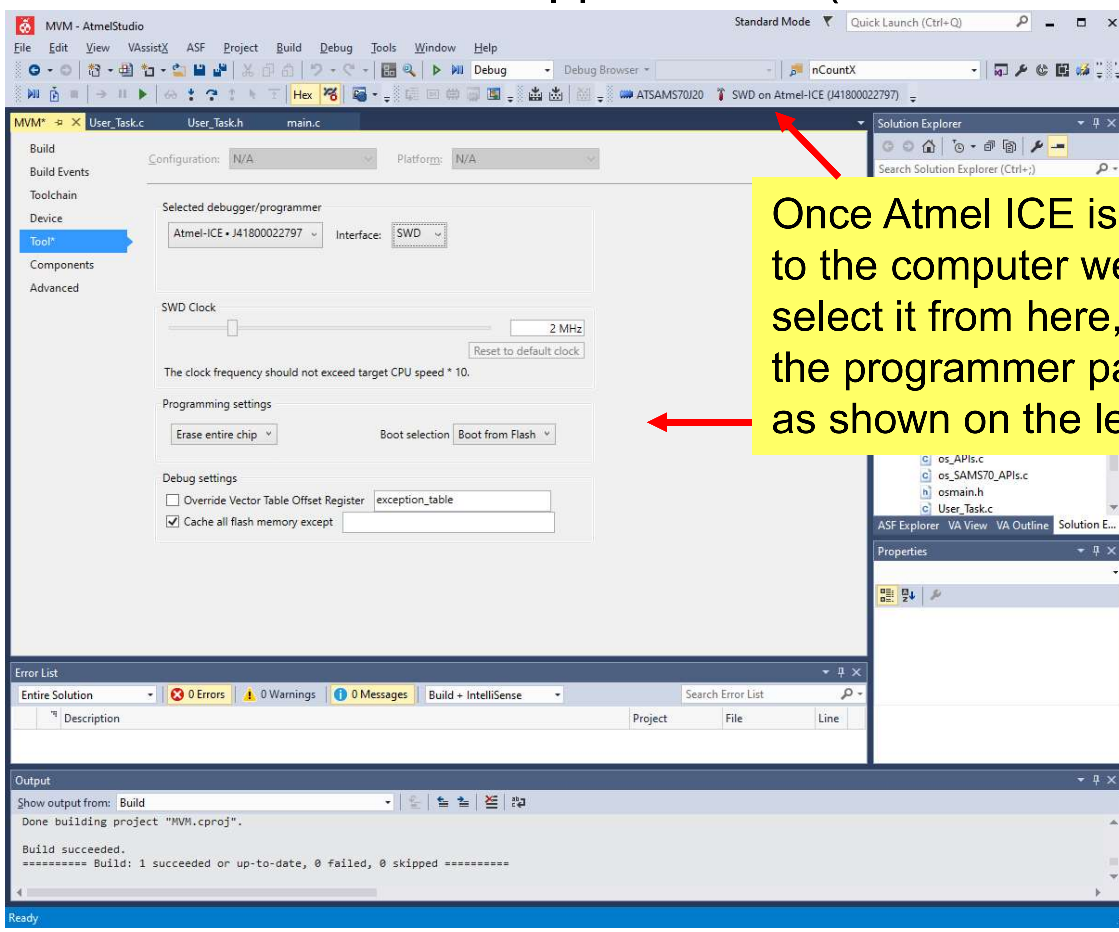


Use this tab to select and setup the programming tool, see next slide.

A success message will be displayed.

Setting Up the Programming Tool – Atmel ICE

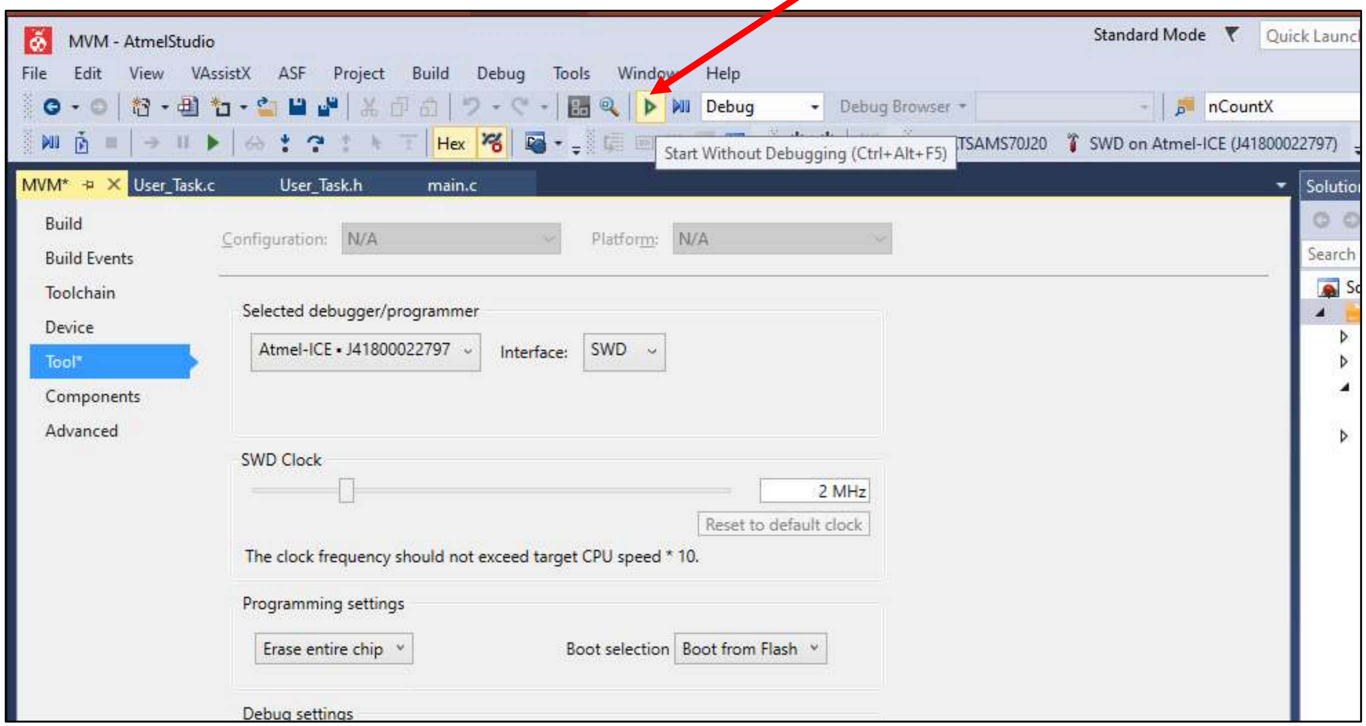
- Now you can load the firmware into the micro-controller with a suitable programmer. Here we are using Atmel ICE, but any programmer compatible with Atmel Studio 7 and support SWD (serial wire debug) mode is fine.



Once Atmel ICE is connected to the computer we can select it from here, and set the programmer parameters as shown on the left.

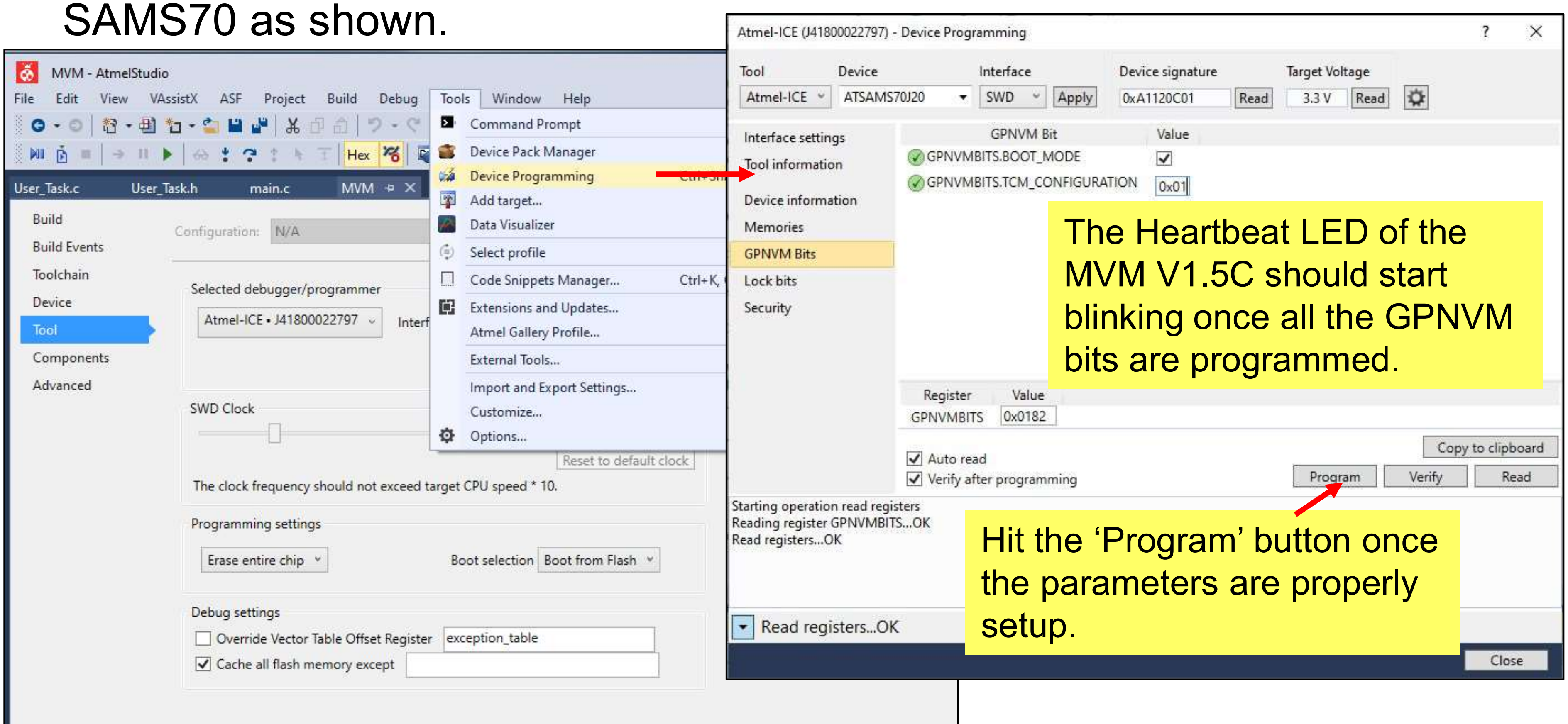
Flashing the Micro-Controller 1

- 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.



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

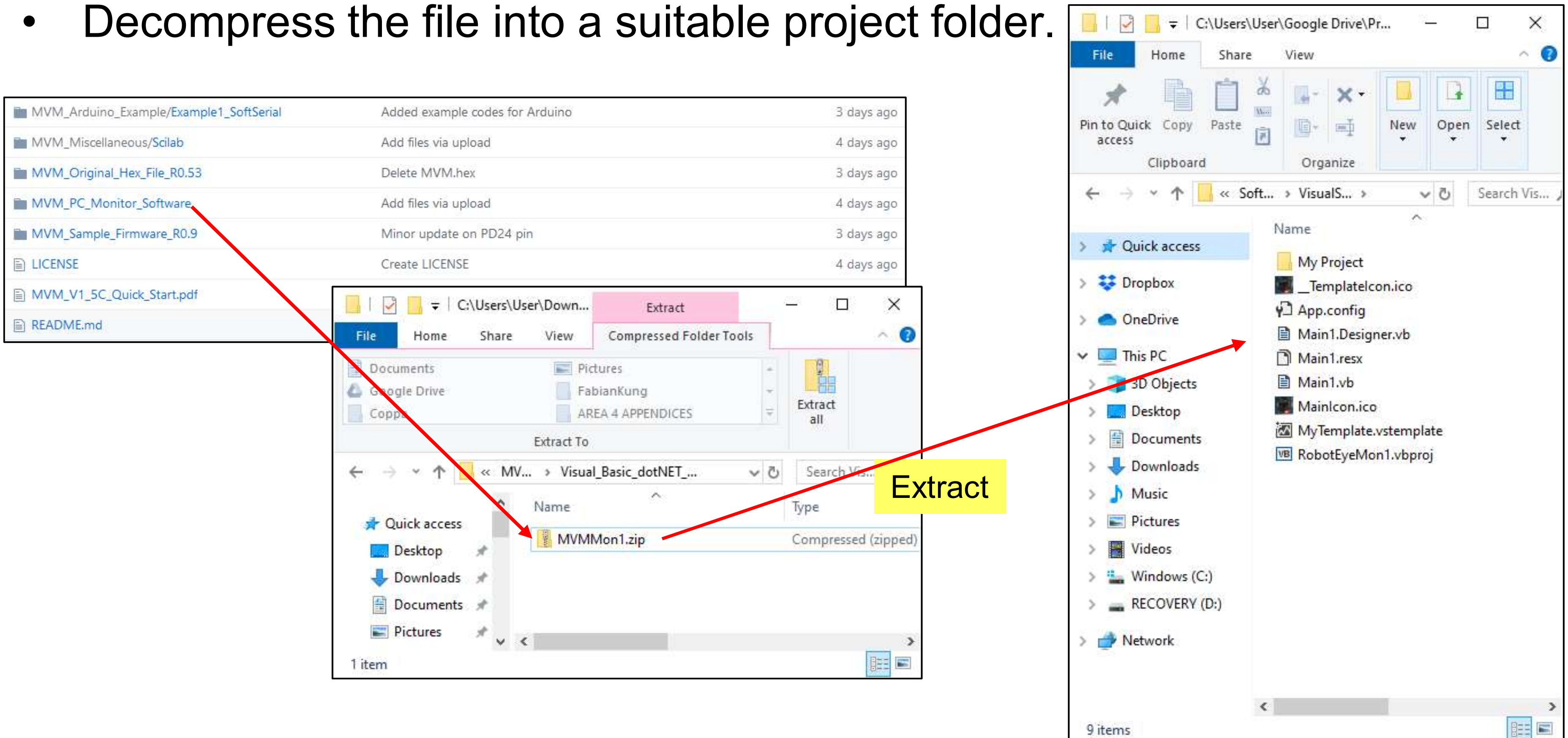
Compiling and Building the Machine Vision Monitor Software

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.

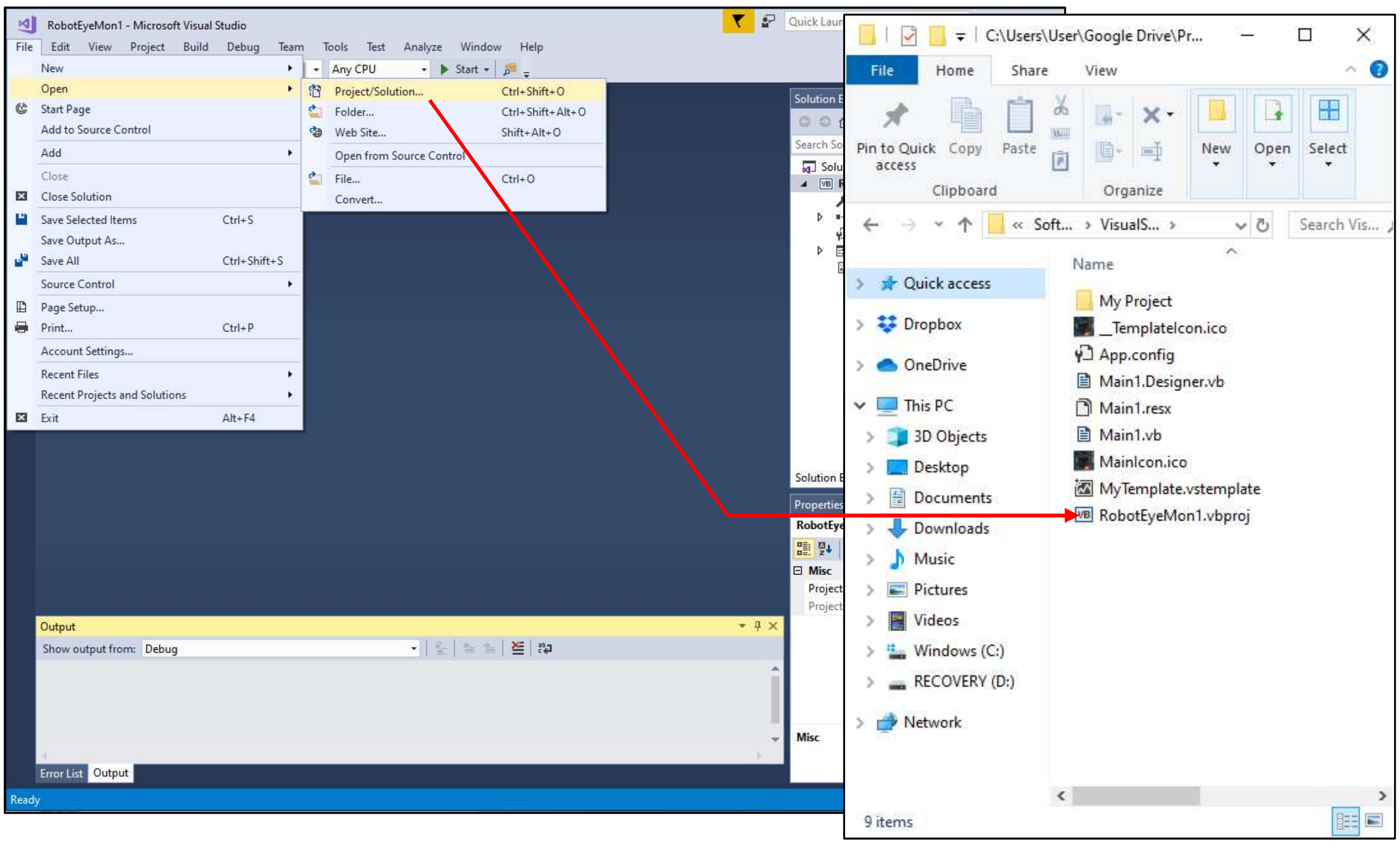
Setting Up Visual Studio Project 1

- In the folder “MVM_PC_Monitor_Software” look for the file MVMMon1.zip.
- Decompress the file into a suitable project folder.



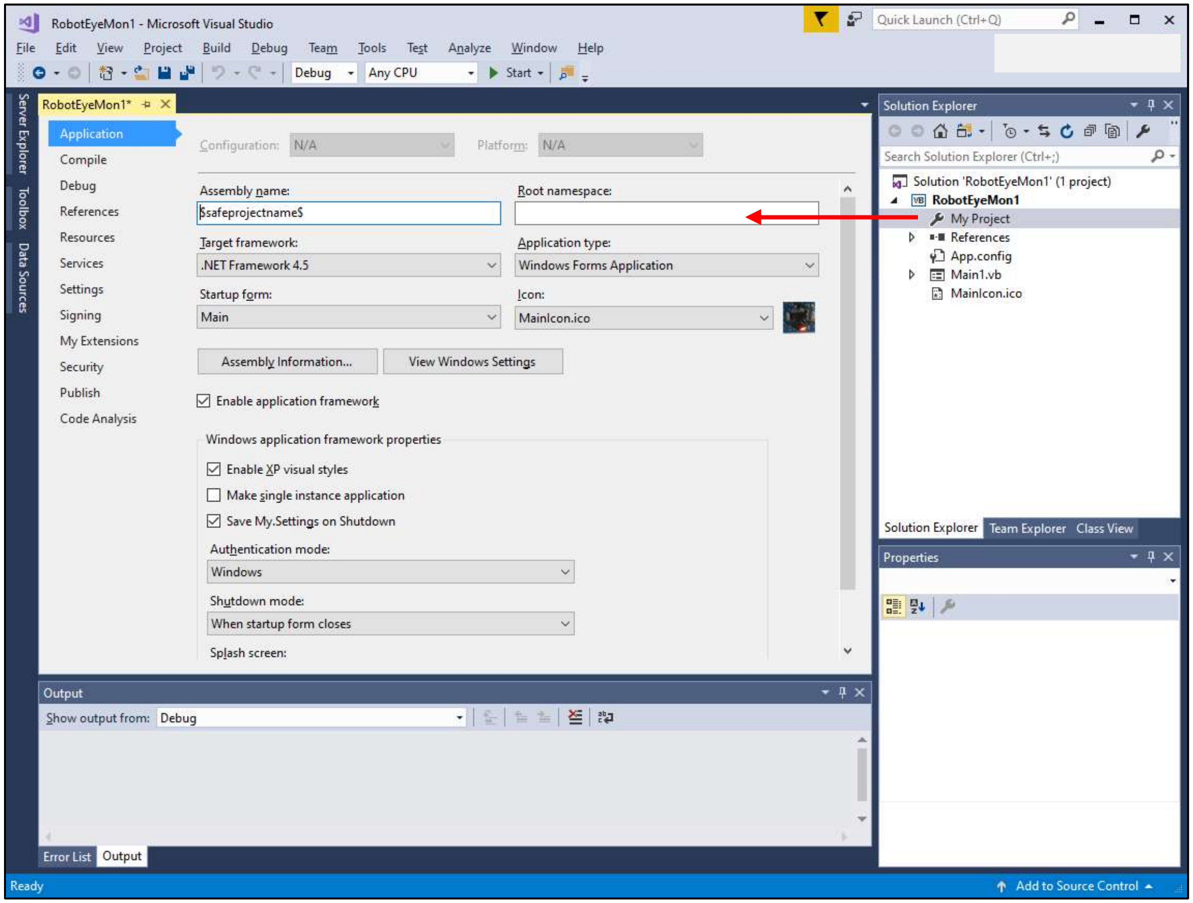
Setting Up Visual Studio Project 2

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



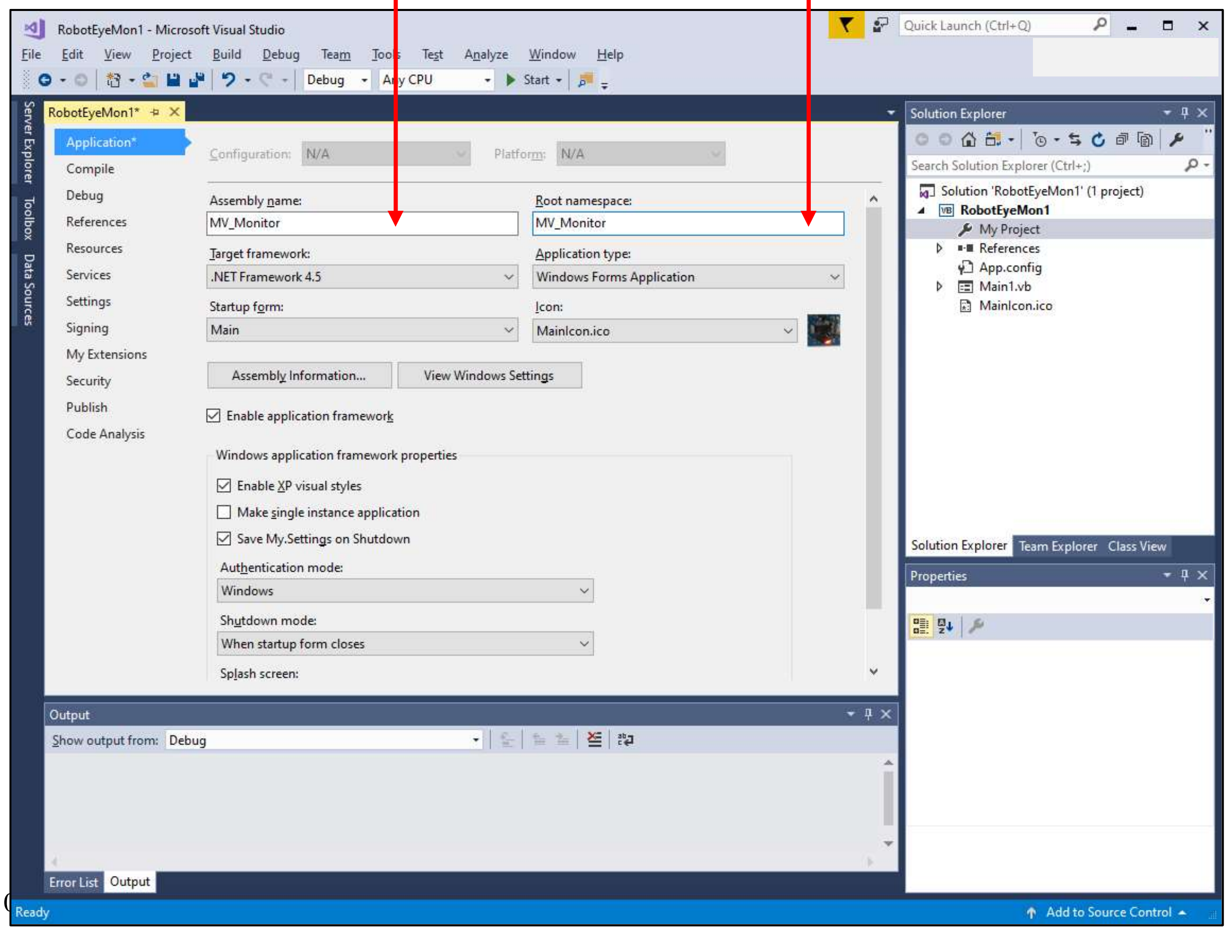
Setting Up Visual Studio Project 3

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



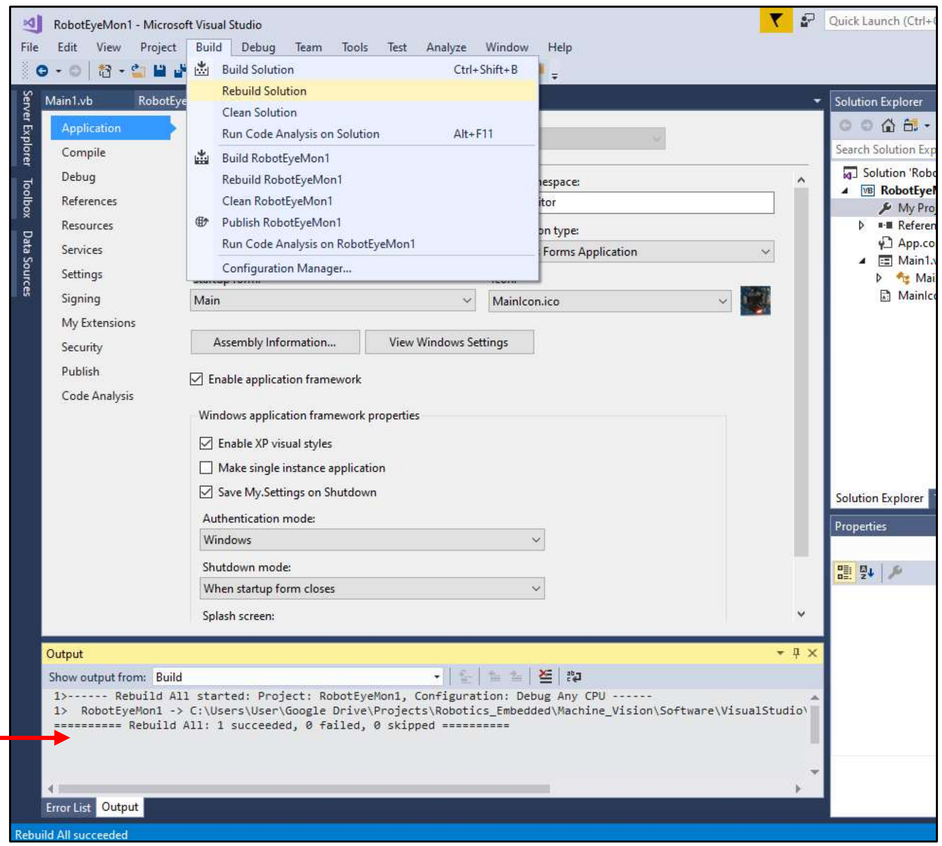
Setting Up Visual Studio Project 4

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



Setting Up Visual Studio Project 5

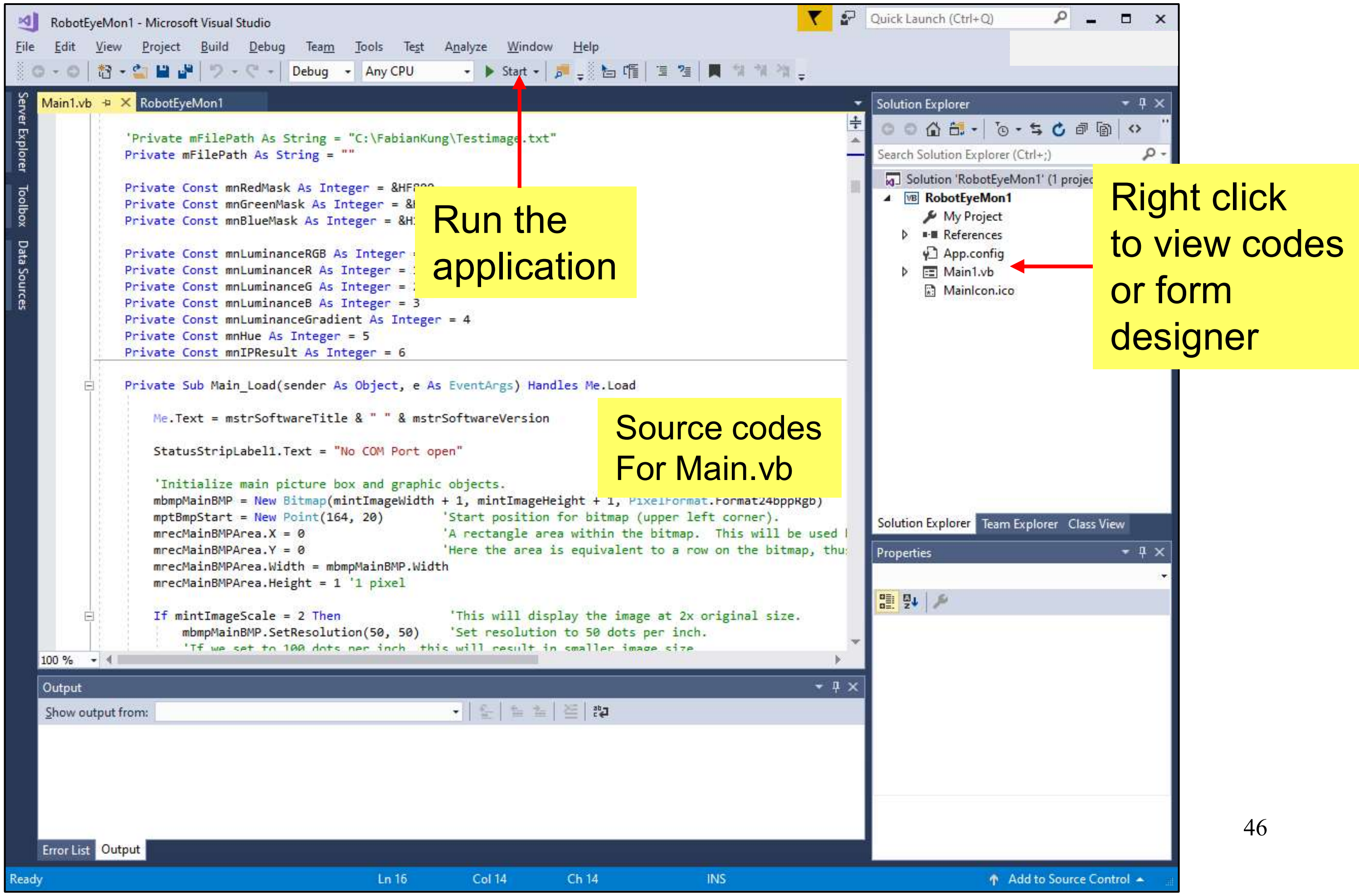
- Now rebuild the project as shown and you should get a success message in the Output window.



Success Message

Setting Up Visual Studio Project 6

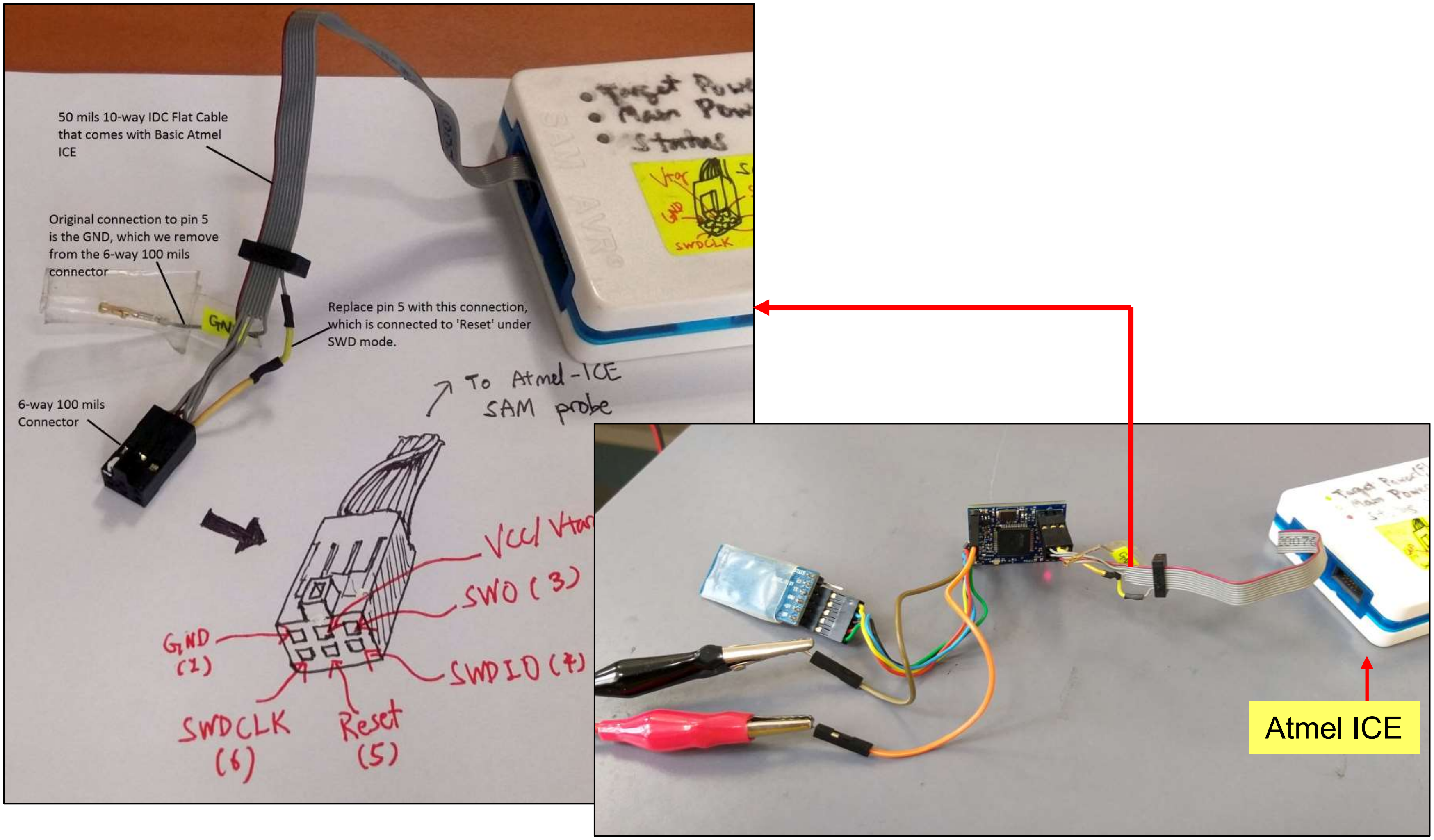
- You can now run the application, view/edit the source code and the main window form.



APPENDIX

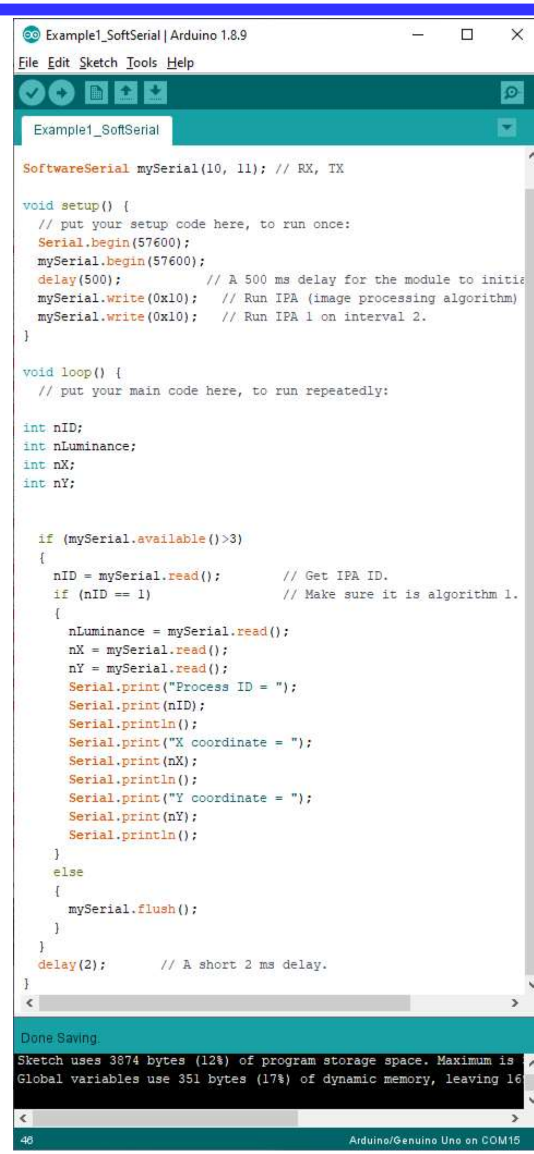
Programmer Connection, Examples and File Export

Connecting Atmel ICE to MVM V1.5C



Example 1 - Using MVM with Arduino Uno and SoftwareSerial Library

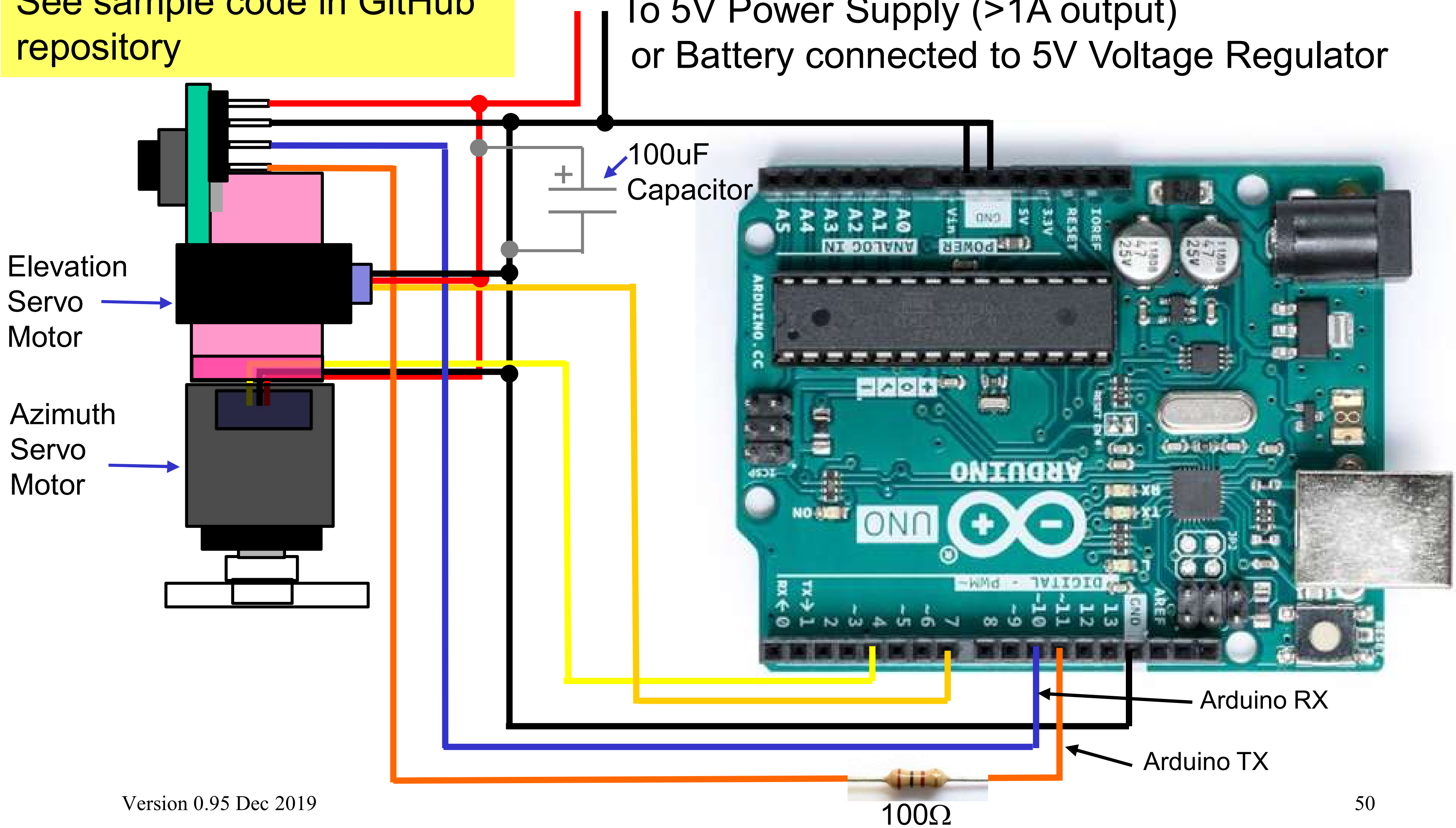
In this code we use SoftwareSerial port to communicate with MVM V1.5C, while the hardware serial is used in conjunction with Serial Terminal for debugging.



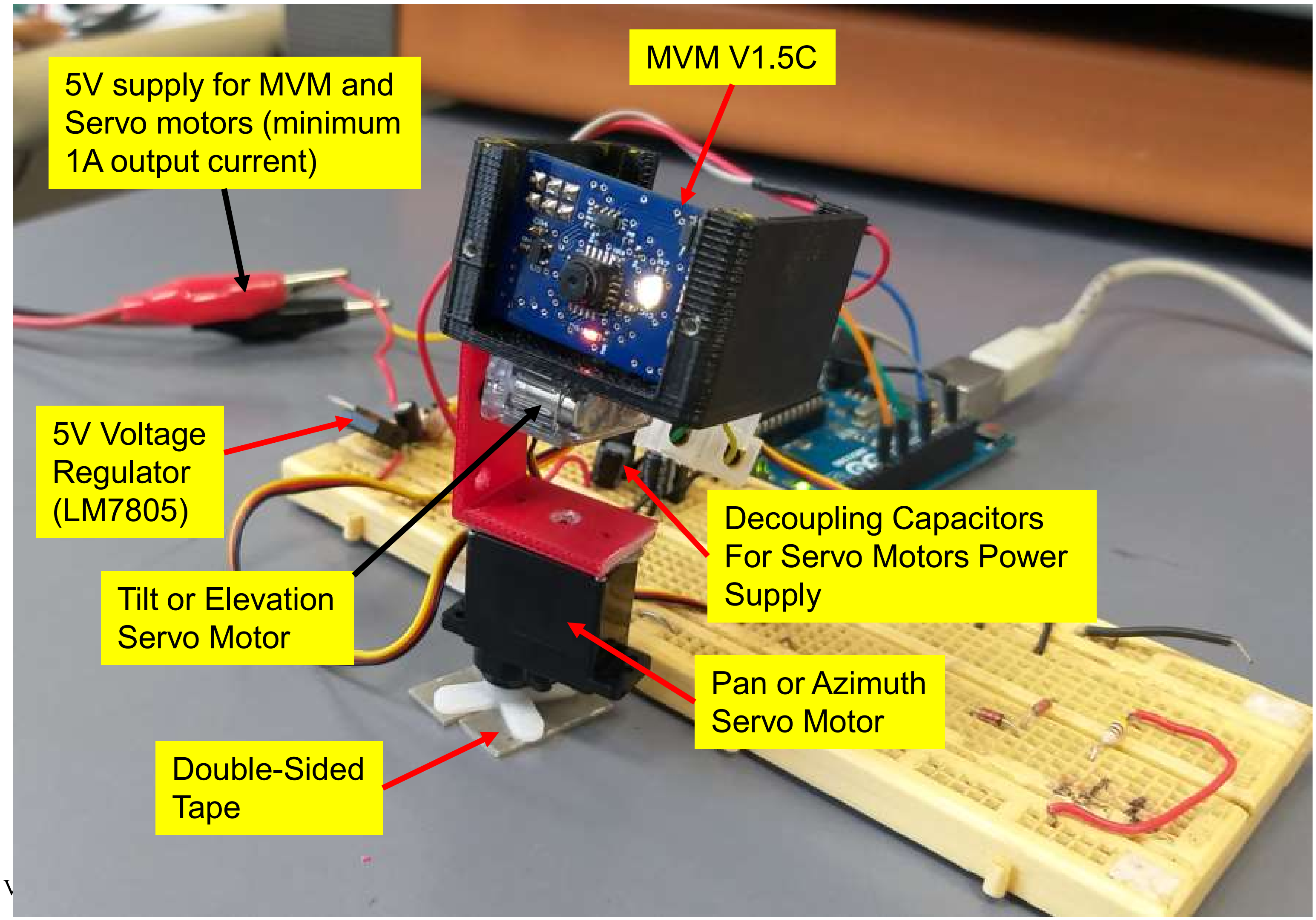

```
SoftwareSerial mySerial(10, 11); // RX, TX

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 initia
  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.
}
```

Example 3 - Color Object Tracking with Arduino Uno

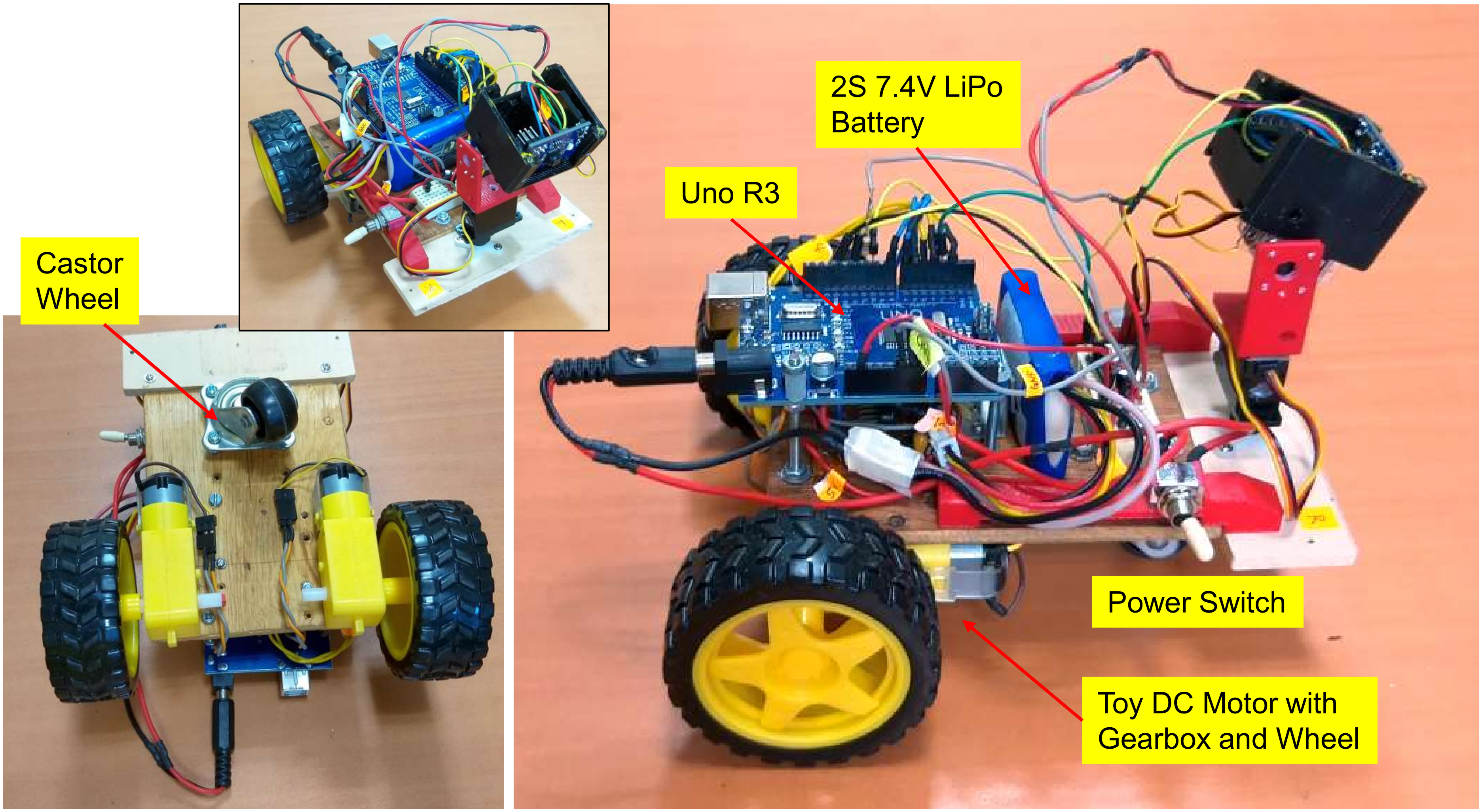
See sample code in GitHub repository



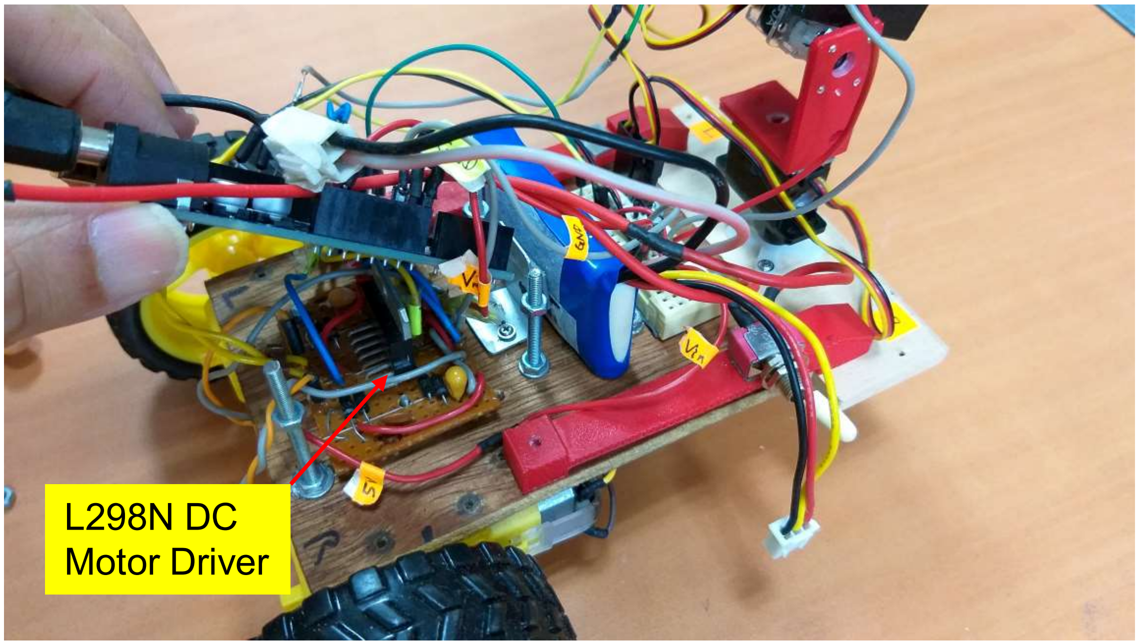
Example 3 - Mechanical Setup



Example 4 - Autonomous Navigation with Arduino Uno Based Robocar



Example 4 - Wiring Information of Robocar



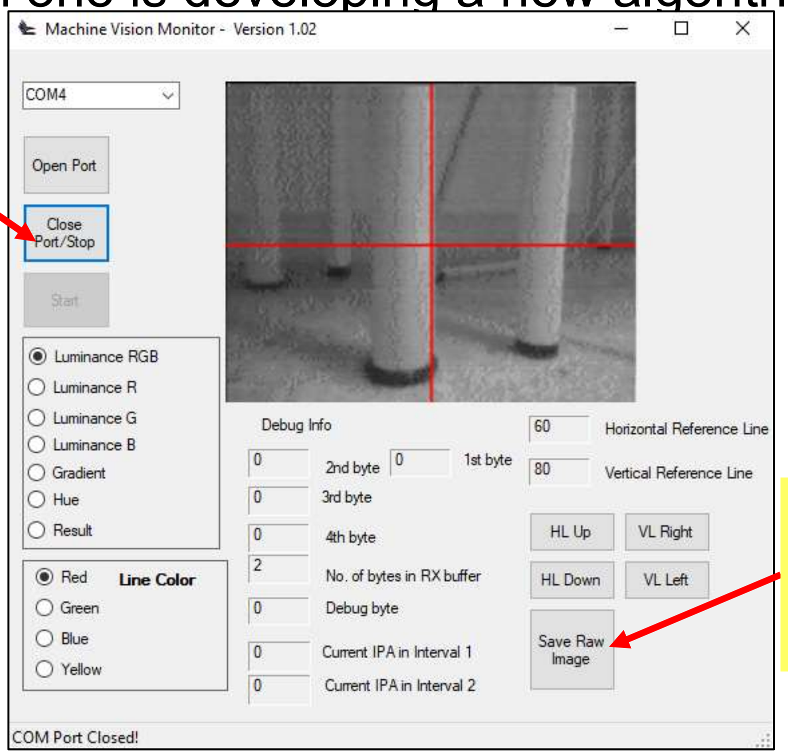
Arduino Uno R3

- 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 100Ω 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 Retriving 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.

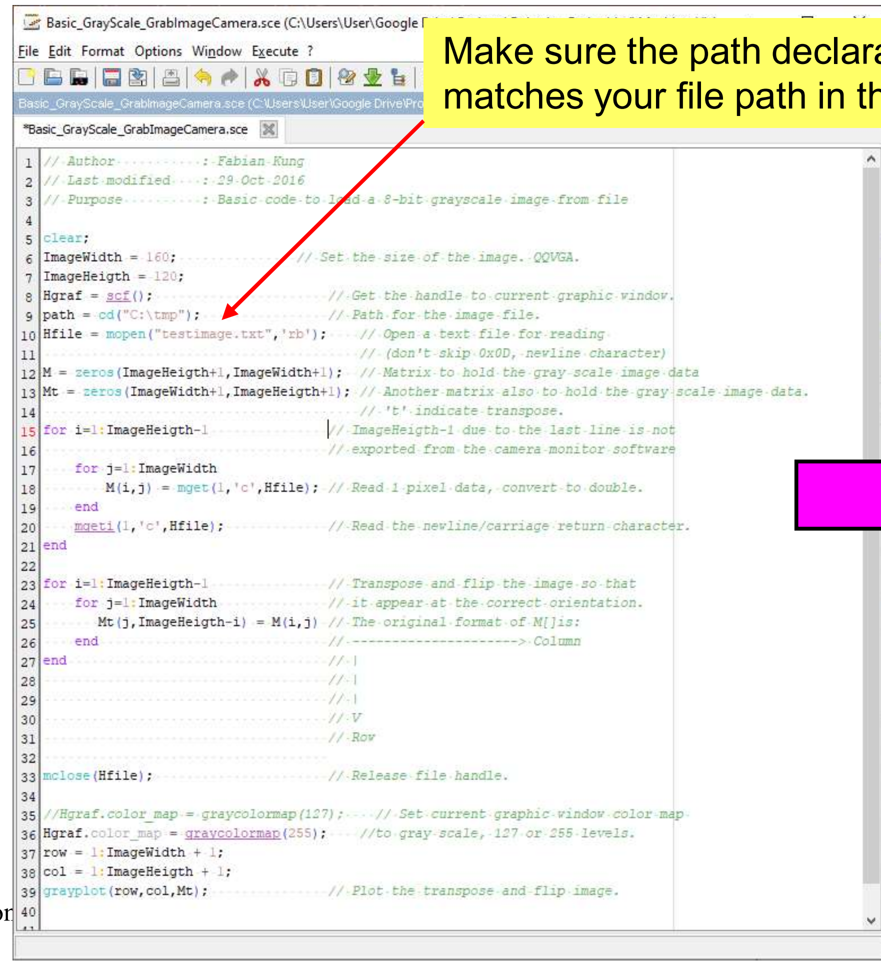
1. Close the COM port to stop streaming image



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.




```
// Author          : Fabian Kung
// 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.
ImageHeigth = 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(ImageHeigth+1,ImageWidth+1);  // Matrix to hold the gray scale image data
Mt = zeros(ImageWidth+1,ImageHeigth+1); // Another matrix also to hold the gray scale image data.
                                        // 't' indicate transpose.
for i=1:ImageHeigth-1           // ImageHeigth-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
    mgeti(1,'c',Hfile);         // Read the newline/carriage return character.
end

for i=1:ImageHeigth-1           // Transpose and flip the image so that
    for j=1:ImageWidth          // it appear at the correct orientation.
       Mt(j,ImageHeigth-i) = M(i,j) // The original format of M[]is:
    end                         // ---------------------> Column
end                             // |
                                // |
                                // |
                                // 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:ImageHeigth + 1;
grayplot(row,col,Mt);           // Plot the transpose and flip image.
```

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