Pong Table

Test Plan & System Integration Status

ECEN5623 Fall 2013

GROUP 7
Bhaumik Bhatt
Edwin R. López
Mayank Bhardwaj
Tejas Joshi
Vinit Vyas

This document describes the **test plan and integration of units in our project so far.** Divided in two parts, the first part describes the current system units and integration status with issues we are running into. Parts of the whole which are functional and the difficulties we faced implementing them are described here. The second part describes unit tests that will be used to validate the different subsystems of the pong table.

---

1. RT patches on the BeagleBone Black Linux:

Upon inspection of the available RT patches for Linux on the BeagleBone Black, we came across patches that can be applied to Linux distributions. Among the multiple options, RTAI and Xenomai looked promising. We went with the previously implemented (thought to be somewhat fail-proof) **LinuxCNC with the Xenomai pre-patched real-time Linux kernel**.

Two methods were followed to write Xenomai real-time kernel image:

1. With the following link to patch-your-own-Xenomai (https://github.com/DrunkenInfant/beaglebone-xenomai), we got stuck with arm-gnueabihf-gcc/g++ tool chain and were not able to do a make operation for creating the images. (Fig. 1)

```
drix@lynx:~/bbb/ipipe-jki$ sudo make ARCH=arm CROSS_COMPILE=arm-linux-g
nueabihf- ZRELADDR=0x80008000 uImage modules
make: arm-linux-gnueabihf-gcc: Command not found
  CHK     include/generated/uapi/linux/version.h
  CHK     include/generated/utsrelease.h
make[1]: `include/generated/mach-types.h' is up to date.
  CC      kernel/bounds.s
/bin/sh: 1: arm-linux-gnueabihf-gcc: not found
make[1]: *** [kernel/bounds.s] Error 127
make: *** [prepare0] Error 2
drix@lynx:~/bbb/ipipe-jki$ cat /usr/bin/arm-linux-gnueabi-gcc
arm-linux-gnueabi-gcc             arm-linux-gnueabi-gcc-nm-4.7
arm-linux-gnueabi-gcc-4.7         arm-linux-gnueabi-gcc-ranlib-4.7
arm-linux-gnueabi-gcc-ar-4.7
drix@lynx:~/bbb/ipipe-jki$ cat /usr/bin/arm-linux-gnueabi-gcc
```

Fig. 1 Attempt to patch Xenomai stuck at arm compiler tool chain

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
      ZRELADDR=0x80008000 uImage modules
```

2. The SD card image to run LinuxCNC with Xenomai real-time kernel was successful and we were able to boot the BBB (http://beagleboard.org/project/MachineKit/)

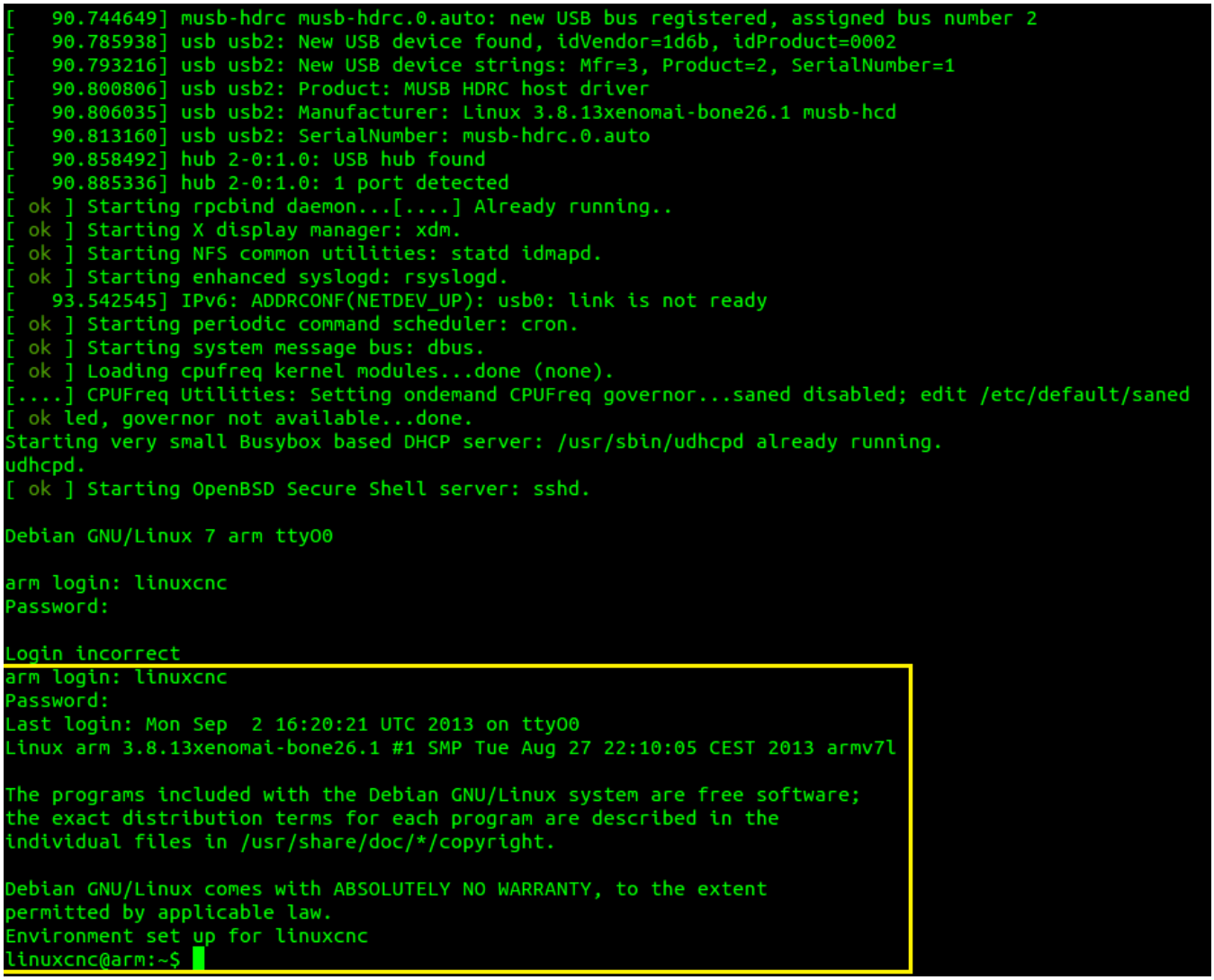


Fig. 2 Pre-patched image of the Debian-Xenomai was loaded onto an SD card with successful boot on the BeagleBone Black

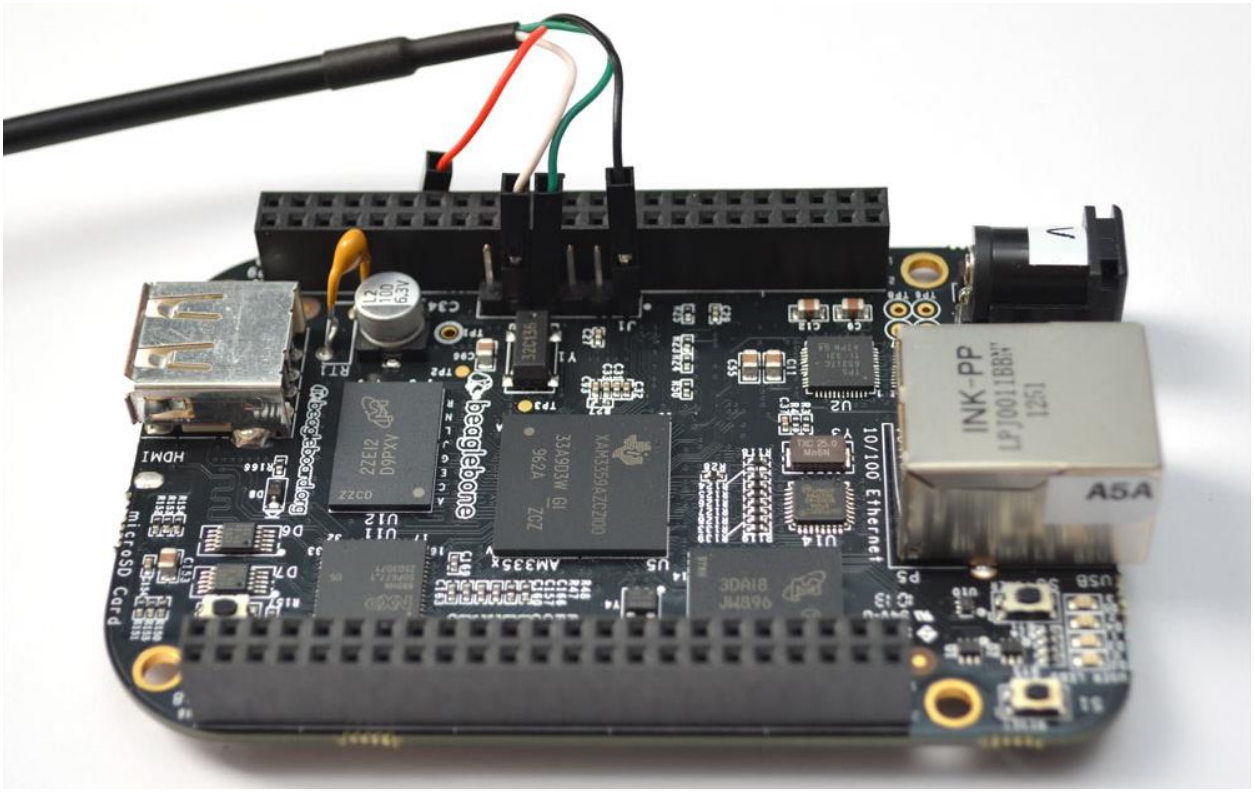


Fig. FTDI Debug used for RX/TX to get to the terminal instead of performing an SSH on the BBB with Xenomai patched RT kernel

Setbacks:

1. Some of our BeagleBone Blacks shipped with a corrupt eMMC and was therefore inaccessible via SSH at all. Had to figure that out and repair that by patching a newer version of the Angstrom Linux distribution on it.

2. Lack of proper documentation for the BeagleBone Black.

3. Lack of proper documentation for which Linux distribution BeagleBone Black works best with.

4. Lack of proper documentation for BeagleBone Black specific tool chains for compiling distributions with the Xenomai patch.

5. Lack of uvcvideo, Ethernet and other required drivers on the LinuxCNC Xenomai distribution, therefore unable to test cameras on it, so far.

2. Video camera interface and frame capture:

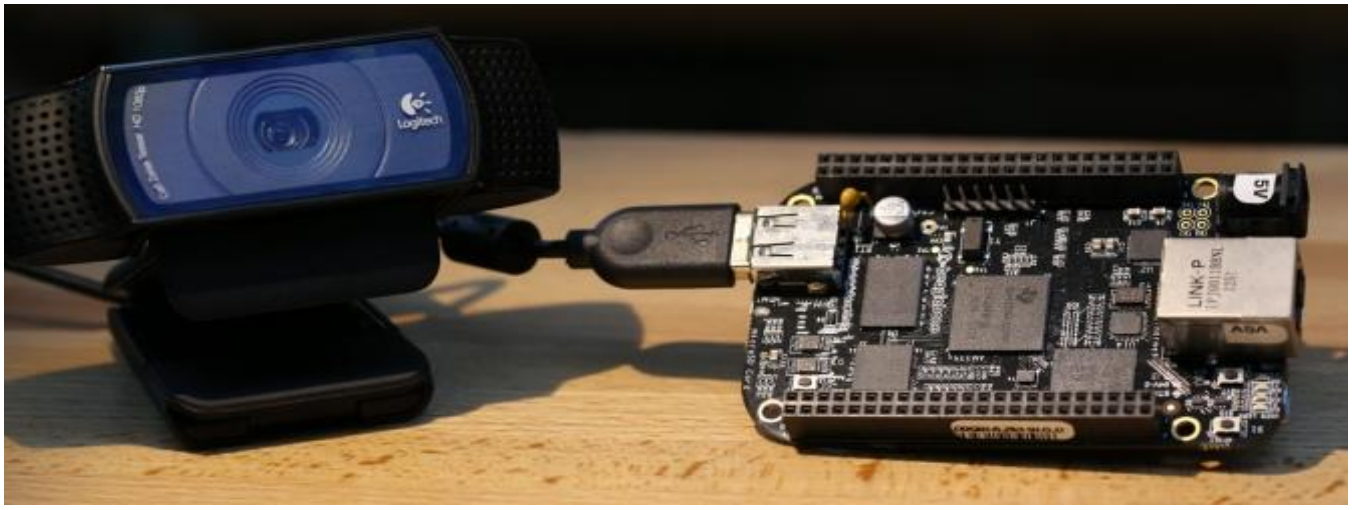


Fig. 3 Logitech USB Camera C920 with BeagleBone Black

PLEASE NOTE: We have used the Angstrom Linux distribution which was shipped with the BeagleBone Black to perform all the unit module tests that follow in this report. Real-time functionality is yet to be added to the code that exists as of now.

Goal: To capture 24 frames per second (12 frames per second per camera)

The frame captures are being implemented using the open source uvcvideo driver. The output files are uncompressed YUYV (4:2:2) format, PPM images. There were multiple tests done to determine timing, bandwidth and latency. We used one camera at first, then branched out to two using a powered USB Hub. The following tests were run when the BBB's CPU governor was set to performance mode (1 GHz).

```
# cpupower frequency-set -g performance
```

Cameras:

1. USB Web Camera from Sparkfun electronics

We were able to capture frames with a resolution of 320x240. But there would exist continual "bad" garbage frames with any resolution beyond that.

2. Logitech C200

Similar to unbranded camera – we observed that frames output at any resolution greater than 320x240 were garbage.

3. Logitech C920

This is the camera we have finalized on. With it, we could capture 1080p resolution images (1920x1080).

Further:

The timing measures showed that we were able to capture 20 HD frames in about 6.7 seconds. We infer that the throughput for USB bandwidth (plus I/O time) should be enough to process 320x240 or 640x480 frames from two cameras simultaneously.

When we connected two cameras at the same time (using a powered USB Hub), and ran two "grabber" programs simultaneously, ***the BBB printed out errors of not having enough USB bus bandwidth.*** This is because both the cameras are being reported as using the entire bandwidth available for the USB port (even when they are actually using much less).

An alternative implementation is to time-slice the captures between two cameras. We are currently reviewing and writing code to do this using POSIX threads and semaphores for frame capture and device synchronization. If this works without much overhead, it would provide the mechanism to alternate between two different frames and use different message queues to signal threads further ahead in the pipeline to process data (filtering/ centroid detection).

Another possibility is of re-opening the uvcvideo driver with a specified quirks value in order to deal with bugs the driver may experience.

Also, we can consider the following workarounds in order to have multiple cameras working simultaneously:

- Disable other devices that allocate periodic transfer bandwidth. This includes keyboards, mice and microphone/speaker.
- Capture video in a compressed format. Many UVC devices support both uncompressed YUV and compressed MJPEG, switching from YUV to MJPEG should reduce bandwidth usage. We are looking in to the option of H.624 compression using openCV libraries. As opposed to lossy, lossless compression seems like a better option.
- Reduce the frame size.
- Connect the cameras to a different USB host controller. This effectively raised the total available bandwidth as each USB host controller can use 480 Mb/s. Extra USB host controllers can be added to the system as cheap PCI cards.

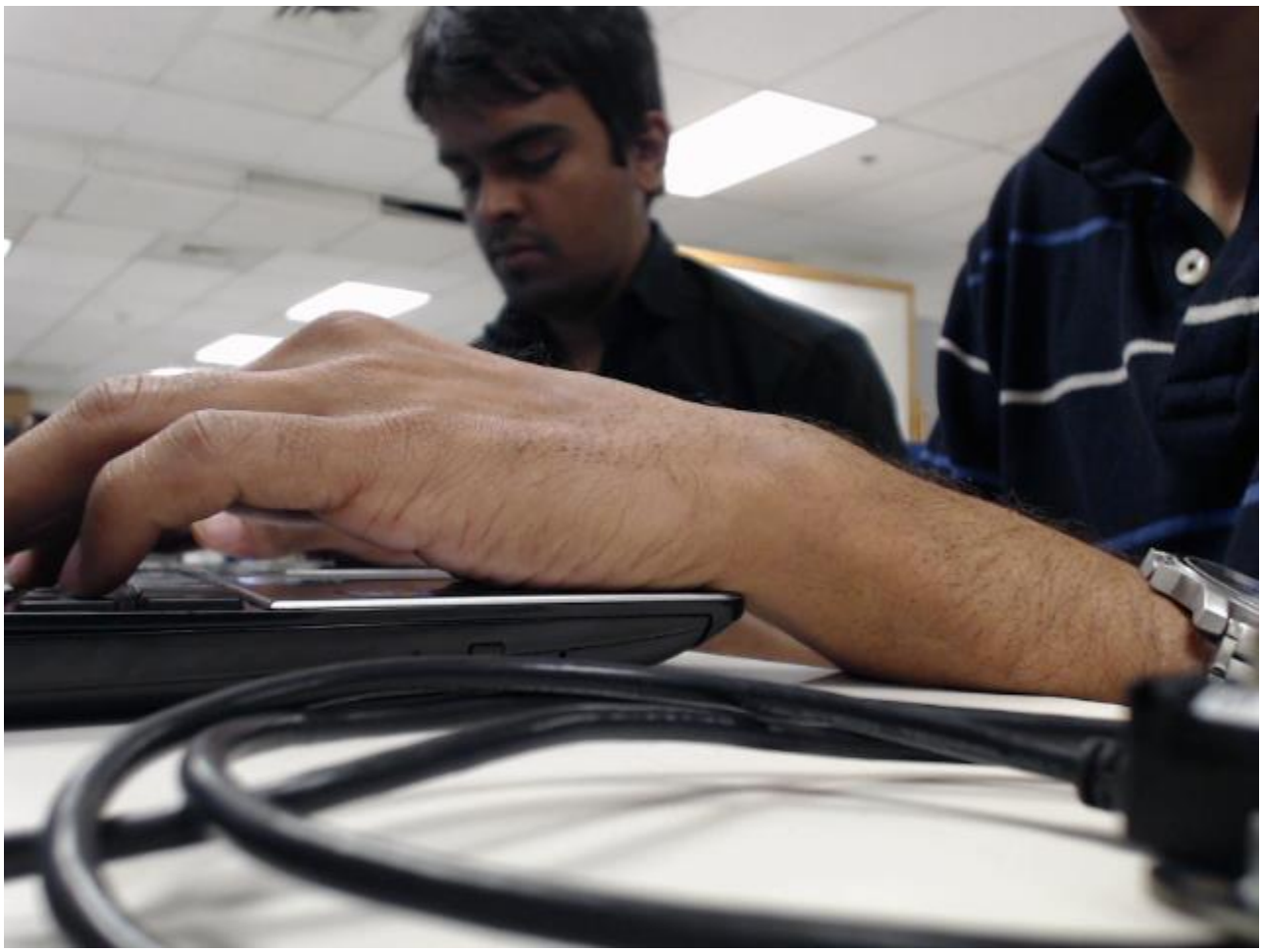
Timing:

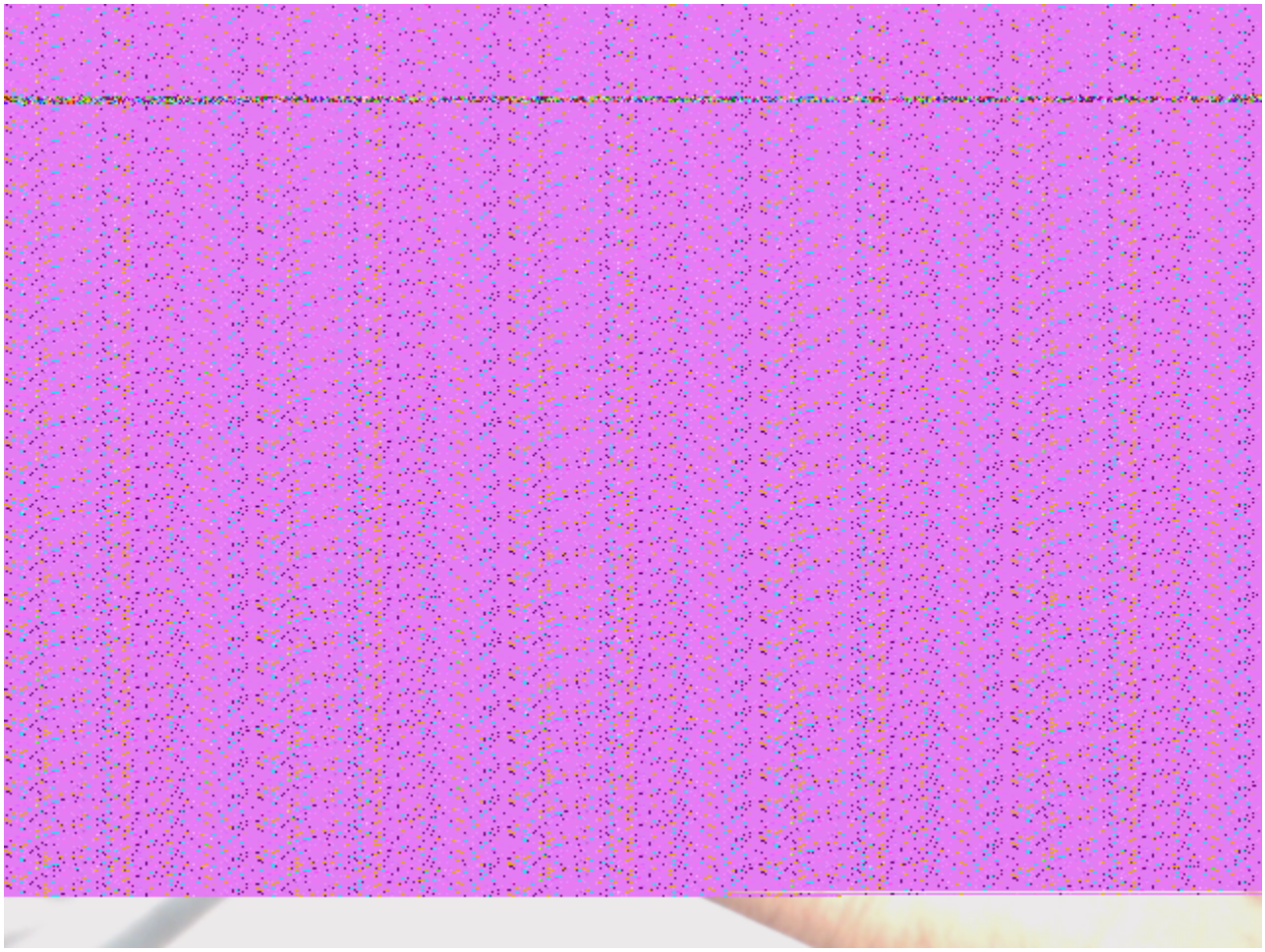
A large part of the time that was measured, was consistent of the I/O write times. This overhead needs to be precisely benchmarked to determine the current performance of the frame capture algorithms. We used a modified frame grabber code from the internet, as well as ffmpeg to capture the frames.

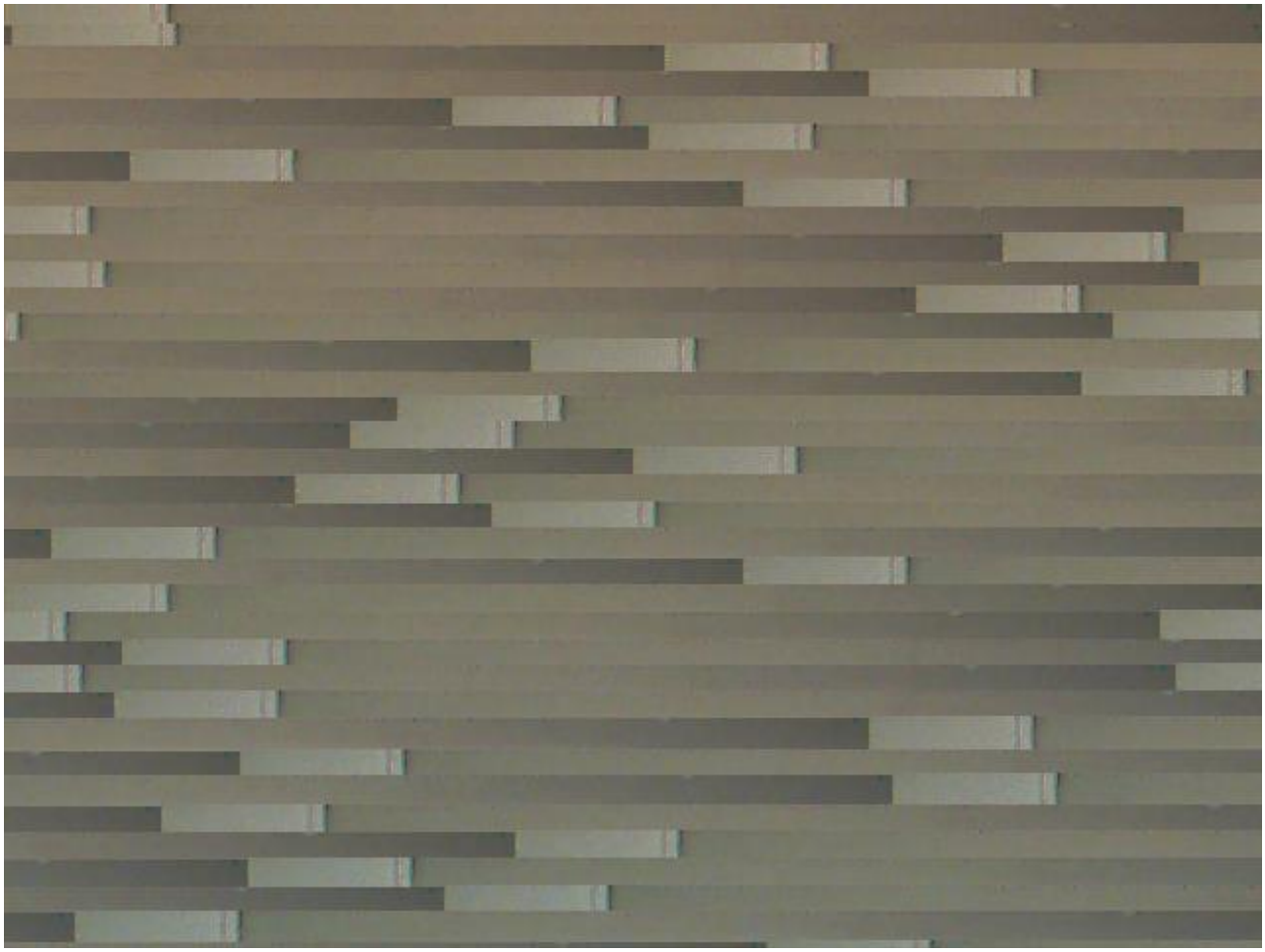
```
ffmpeg -f video4linux2 -s 640x480 -vframes 12 -i /dev/video0 -f
image 2 image%2d.ppm
```

Result was that ffmpeg was faster to return with the complete PPM (RGB) image than the grabber code. However, the results with ffmpeg were not consistent—the first frame captured had issues with the color saturation, one out of ten occasional bad frames on 640x480 resolution. ***We are leaning towards using our own implementation using v4l driver instead of ffmpeg directly***. Additionally, there is a known issue with uvcvideo driver, which mentions reliability/performance issues if a frame rate of less than 30 fps is used for a resolution below 320x240. We are not yet sure if this is the reason the first image was highly saturated, or if it was just a camera/ encoder issue.

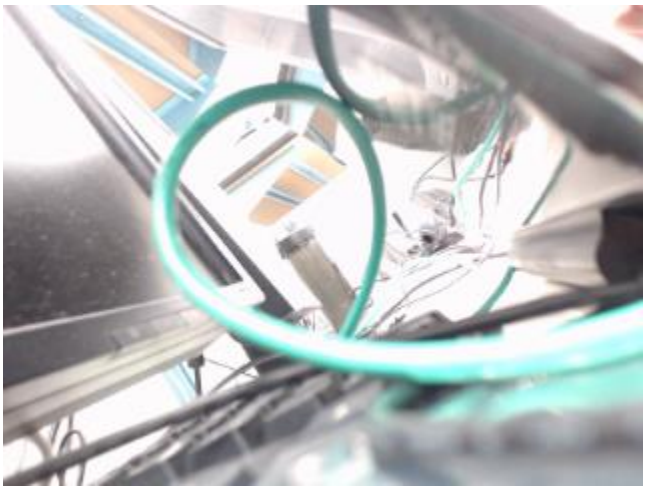
PPM file Captures:







320x240 resolution images:



Compression:

We have looked in to usage of H.264 codec by changing format on Video4Linux2 to H.264 and size was kept at 640x480 instead of 1920x1080 as we need speedy captures.

We are yet to completely test this and gain results. We ran in to some problems with openCV and H.264 and therefore, could not complete this.

OpenCV and color filter:

We are looking forward to set up openCV on the Angstrom Linux distribution of our BBB. We have looked in to the kind of image color detection and sample code which we will use:

- To load an image:

```
Mat image;

image = imread(argv[1], CV_LOAD_IMAGE_COLOR);   // Read the file
```

- Determine color intensities in a 3 channel image with BGR color ordering (the default format returned by imread):

```
Vec3b intensity = img.at<Vec3b>(y, x);

uchar blue = intensity.val[0];

uchar green = intensity.val[1];

uchar red = intensity.val[2];
```

- We can use the following to isolate colors (providing a basic template):

```
Mat redColorOnly;

inRange(src, Scalar(lowBlue, lowGreen, lowRed), Scalar(highBlue, highGreen, highRed), redColorOnly);

detectSquares(redColorOnly);
```

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <iostream>

using namespace std;
using namespace cv;

Mat redFilter(const Mat& src)
{
  assert(src.type() == CV_8UC3);

  Mat redOnly;
  inRange(src, Scalar(0, 0, 0), Scalar(0, 0, 255), redOnly);

  return redOnly;
}

int main(int argc, char** argv)
{
  Mat input = imread("colored_squares.png");

  imshow("input", input);
  waitKey();

  Mat redOnly = redFilter(input);

  imshow("redOnly", redOnly);
  waitKey();

  // detect squares after filtering...
  return 0;
}
```

Color detection can be aided by HSV (Hue, Saturation and Value) software (If we need, we can use open source ColorWheelHSV for test and surety purposes in order to confirm our values) but mostly, we will work on our own algorithms for it.

Blue matrix

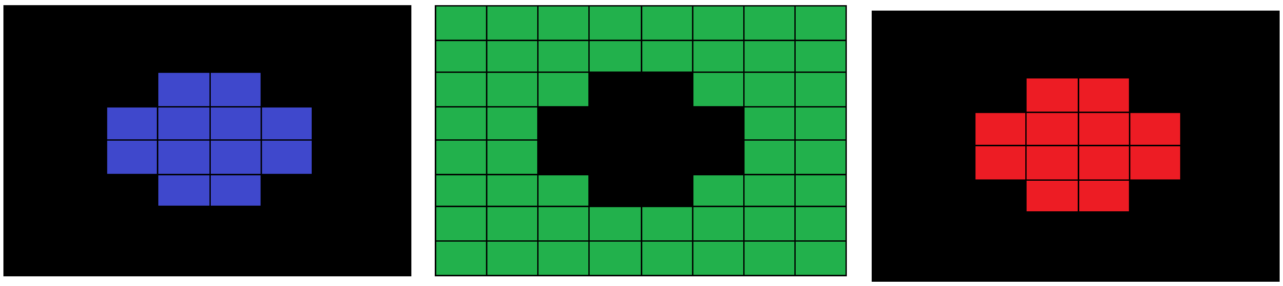
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	255	255	0	0	0
0	0	255	255	255	255	0	0
0	0	255	255	255	255	0	0
0	0	0	255	255	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Green matrix

255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	0	0	255	255	255
255	255	0	0	0	0	255	255
255	255	0	0	0	0	255	255
255	255	255	0	0	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255

Red matrix

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	255	255	0	0	0
0	0	255	255	255	255	0	0
0	0	255	255	255	255	0	0
0	0	0	255	255	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0



Resultant Image

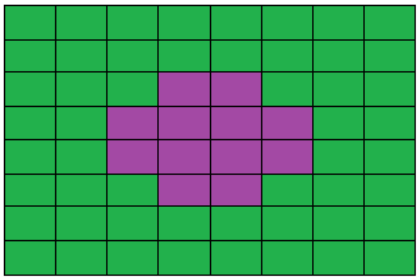
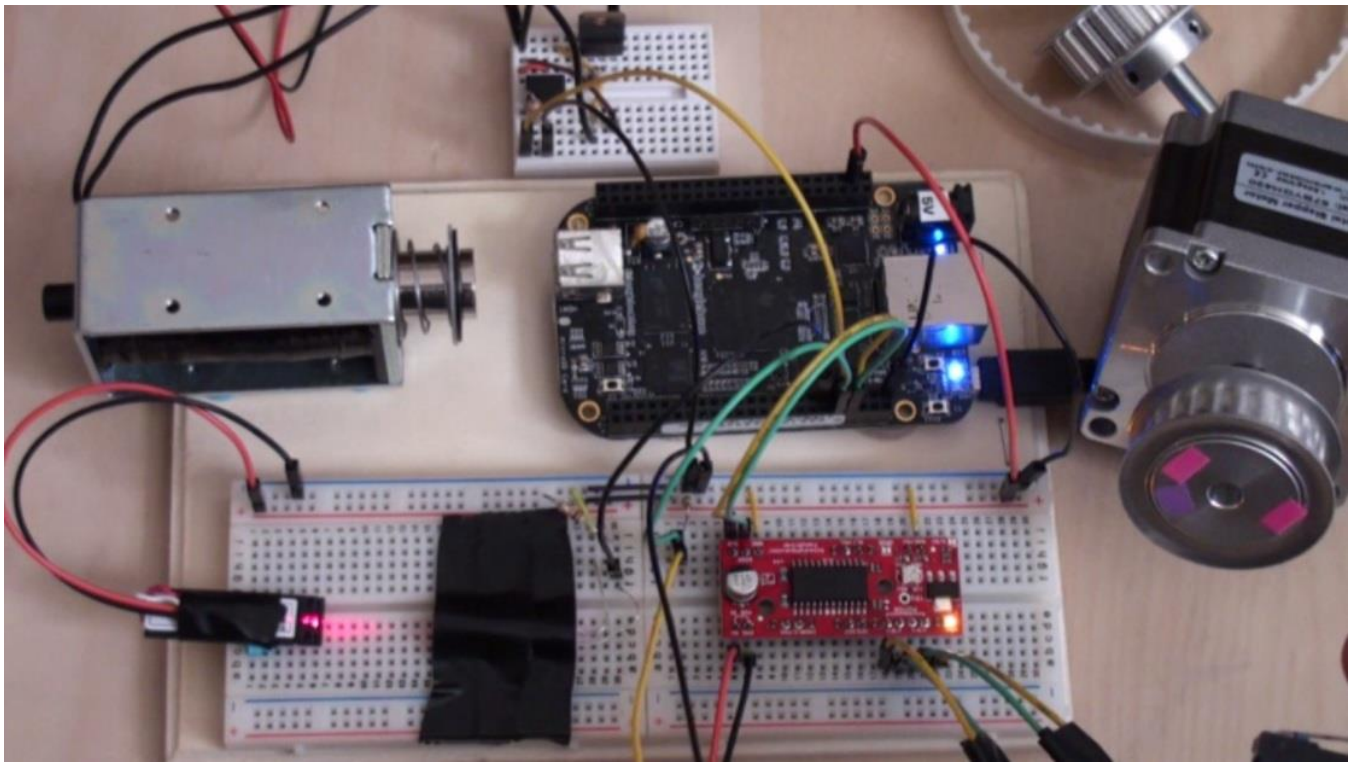


Fig. 4 Color BGR representation sample

3. Hardware implementation:

Overall:



We have put together a video file to explain proof of concept by testing out most of the mechanical and electrical components interfaced to the BBB achieving satisfactory functionality.

1. Stepper Motor:

Implementation Status: Working. Ready to accept inputs & process data to rotate the motor in steps. Further improvement: Implementation of micro-steps.

Board used for implementation:

1. BeagleBone Black with Angstrom Linux (Final Testing) (http://beagleboard.org/static/Docs/Hardware/BeagleBone_Black_DOCS.zip)
2. onboard, an AVR ATmega64 Microcontroller based board (Initial Testing) (http://uniboard.googlecode.com/files/uNiBoardv1_1_reference_manual_rev1.pdf)

Components:

1. EasyDriver Stepper Motor Driver : ROB-10267 (http://www.sparkfun.com/products/10267)

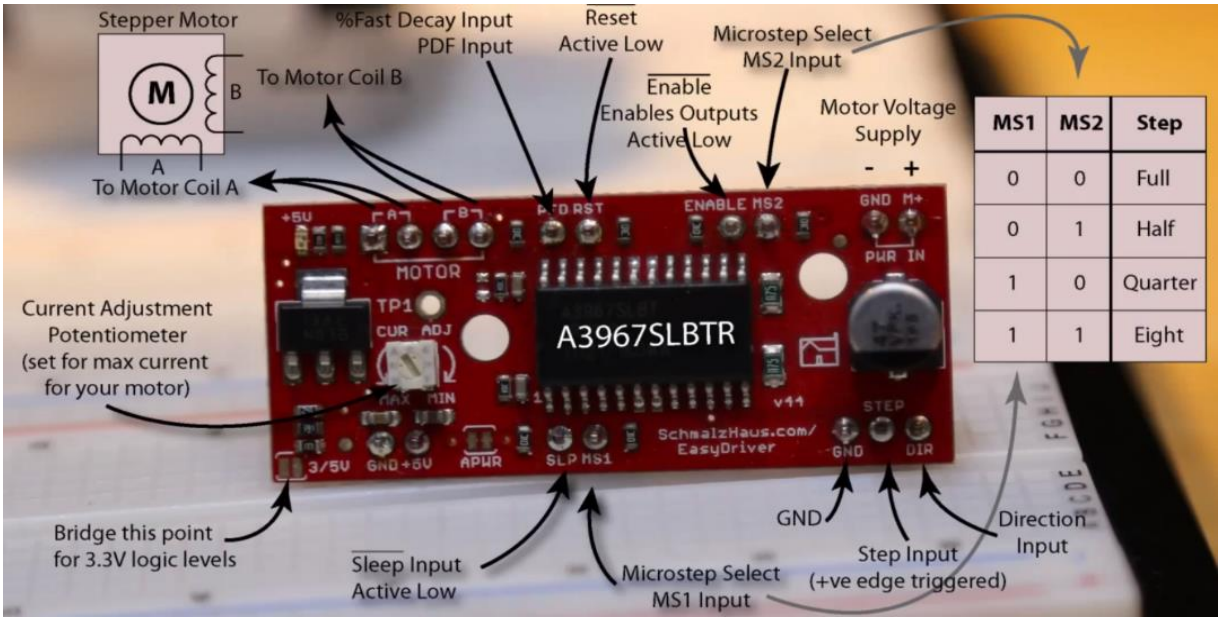
   a. Features:

   -±750 mA, 30 V Output Rating
   -Satlington® Sink Drivers
   -Automatic Current-Decay Mode Detection/Selection
   -3.0 V to 5.5 V Logic Supply Voltage Range
   -Mixed, Fast, and Slow Current-Decay Modes
   -Internal UVLO and Thermal Shutdown Circuitry
   -Crossover-Current Protection

   b. Overview:

   The EasyDriver is a simple to use stepper motor driver, compatible with anything that can output a digital 0 to 5V pulse (or 0 to 3.3V pulse if you solder SJ2 closed on the EasyDriver). EasyDriver requires a 7V to 30V supply to power the motor and can power any voltage of stepper motor. The EasyDriver has an on board voltage regulator for the digital interface that can be set to 5V or 3.3V. Connect a 4-wire stepper motor and a microcontroller and you've got precision motor control! EasyDriver drives bi-polar motors, and motors wired as bi-polar.

   c. Connections: (Image source: http://derekmolloy.ie/)



2. Stepper Motor : 6-wire Sanyo Denki 103H7121-5640 (http://www.newark.com/sanyo-denki-sanmotion/103h7121-5640/stepper-motor/dp/34R9663)

A. Features

-Coil Type: Bipolar
-Current Rating: 1A
-No. of Phases: Two
-Resistance: 4.3ohm
-Inductance: 14.5mH
-Inertia: 0.55oz-in
-Leaded Process Compatible: No
-NEMA Size: 23
-Peak Reflow Compatible (260 C): No
-Supply Voltage: 24VDC
-RoHS Compliant: Yes

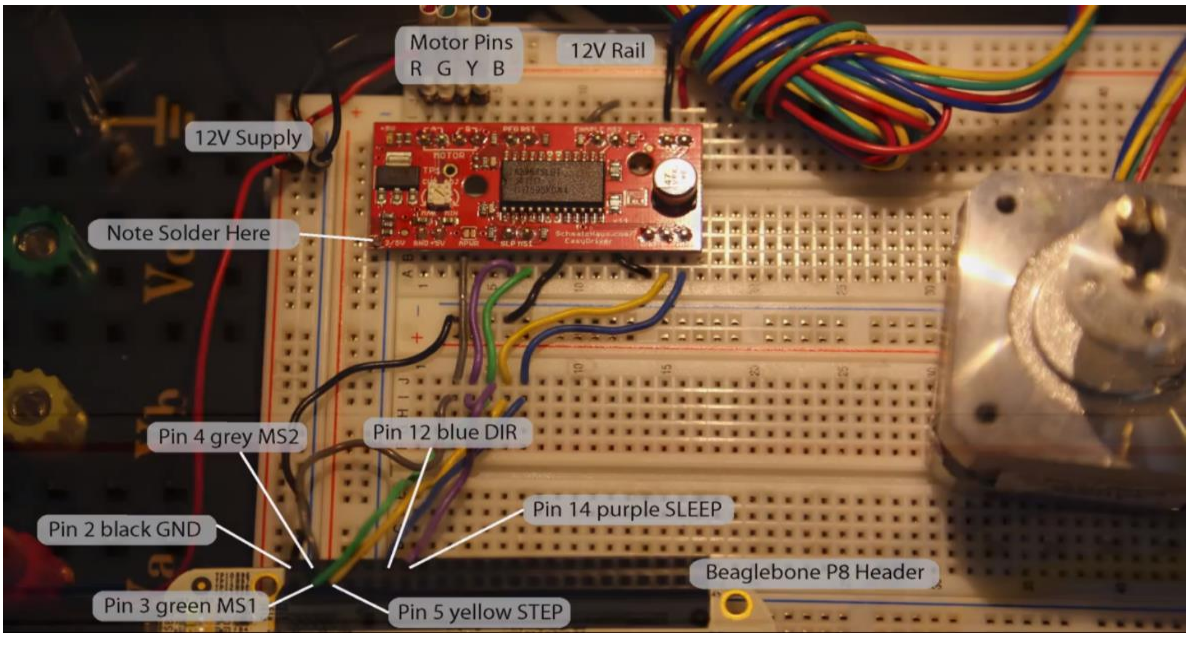
B. Overview

Driver and motor are now integrated into a single unit. A driver incorporating a motion control function needed for driving a motor and a 2-phase stepping motor were integrated into a single unit for enabling a more compact installation space and less wiring.

Three types of operation modes can be selected to match the specific application:

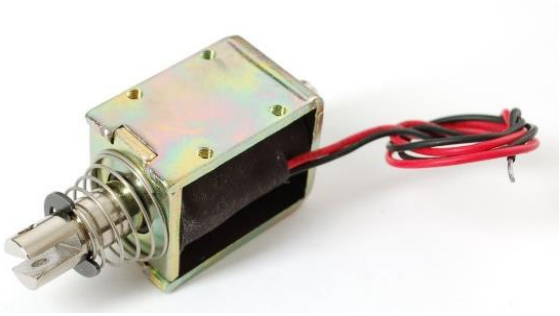
1. Control by command pulses
2. Program control by general-purpose I/O (Parallel)
3. Compliant with RS-485, half-duplex asynchronous communication

C. Image



2. Solenoid:

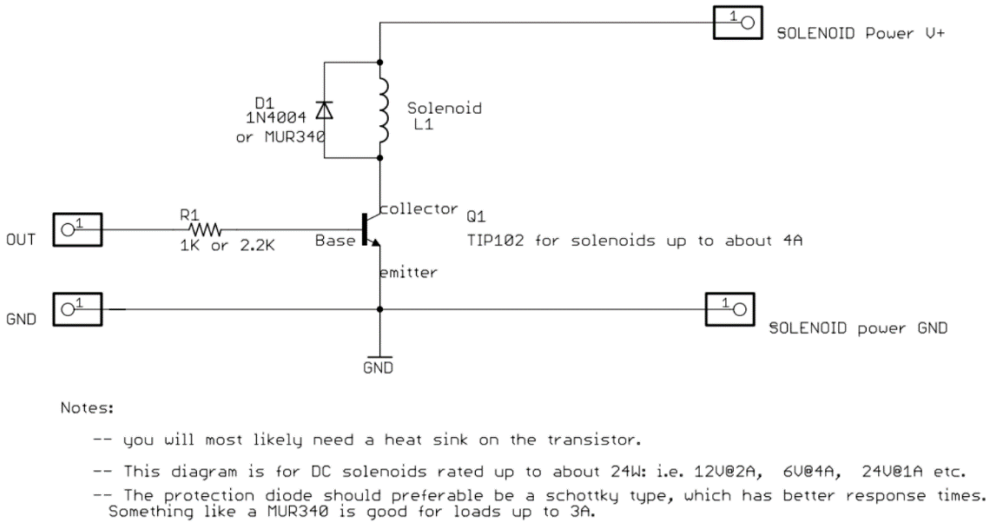
A. Features:



Solenoids are electromagnets, made of a big coil of copper wire with an armature (a slug of metal) in the middle. When the coil is energized, the slug is pulled into the center of the coil. This makes the solenoid able to pull (from one end) or push (from the other).

This solenoid in particular comes with a 40mm long body and a 'captive' armature with a return spring. This means that when activated with up to 24VDC, the solenoid moves and then the voltage is removed it springs back to the original position, which is quite handy. Many lower cost solenoids are only push type or only pull type and may not have a captive armature (it'll fall out!) or don't have a return spring. This solenoid has 4xM3 (metric) threaded holes in the bottom for easy mounting.

B. Basic Diagram:



4. Unit Tests

This section lists the test cases and checklists used to test each subsystem individually. Subsystem tests are carried out by a suite of command line programs. The information reported by each command line program is used to both calibrate the system and detect when any particular subsystem is failing of producing erroneous results.

Description of these programs are presented at the end of this section. Some commands test the table as a whole, while others take an argument, *-a* or *-b*, specifying which half of the table is to be tested: either half *A* or half *B*.

1. Video Hardware Test

- ☐ When connecting both cameras to the board use a **powered** USB hub to
- ☐ Use the 5V power adapter to enable the processor higher speed
- ☐ Verify **video0** and **video1** show on **/dev**
- ☐ Verify that each video capture device can be individually opened and frames can be captured successfully. Use the **v4l2grab** to capture JPEG images.
- ☐ Verify the two video capture devices can be opened simultaneously and frames are captured successfully from each. Run **grab0** and **grab1** simultaneously in separate window shells and verify the *.ppm files are created successfully for each process.

2. Camera Installation Test

- ☐ Verify each camera alignment and adjust the camera support arm up or down until its respective half of the table covers the entire video frame (from the center line to the paddle line). Use the **v4l2grab** to capture JPEG images to monitor the adjustment.
- ☐ Ensure the table surface does not reflect light from celling directly into the camera. Position the table to avoid hot spots.
- ☐ Adjust camera color saturation, contrast, and brightness for the room lighting conditions using v4l2-ctl. Ensure the red puck and green markers on top of the paddle are clear. Use the **v4l2grab** to capture JPEG images to monitor the adjustment.

3. Limit Switch Test

- ☐ Run the **testlimit** command to verify all four limit switches are working (two on each paddle). Press each switch manually and verify the corresponding message for that switch is printed in the shell console.

If a message is not printed in the console when a switch is pressed the cause could be:

- A wire is disconnected, check bread board connections.
- Device Tree Overlay was lost and GPIO pin was reverted to factory default. Re-apply tree overlay.

4. Motor Test

- ☐ Power up the motors
- ☐ Run the **testmotor -a** and **testmotor -b** commands to test motion on each paddle. The motors will slowly move the paddle to the left until it reaches the limit switch. Then the paddle will move to the right until it hit the opposite limit switch.

If the motors do not move the cause could be:

- Power to the **EasyDriver** is off.
- A wire is disconnected, check bread board connections.
- Device Tree Overlay was lost and GPIO pin was reverted to factory default. Re-apply tree overlay.

5. Laser Sensor Test

- ☐ Power up the laser modules.
- ☐ Run the **testlaser -a** and **testlaser -b** commands and verify each laser sensor is working (one at each side of the table). Interrupt the laser beam with an object and verify a message is printed in the shell console indicating the beam interruption was detected.

If a message is not printed in the console when the laser is blocked the cause could be:

- Wire in the receptor sensor is disconnected, check bread board connections.
- Laser beam is misaligned.

- Device Tree Overlay was lost and GPIO pin was reverted to factory default. Re-apply tree overlay.

6. Kicker Sensor Test

☐ Power up the solenoids

Run the **testkicker** command and verify each paddle kicker is working. This command will fire the kicker three times for each paddle (fist paddle A and then paddle B).

If the kicker does not fire the cause could be:

- Solenoid is not receiving power, check bread board connections.
- Device Tree Overlay was lost and GPIO pin was reverted to factory default. Re-apply tree overlay.

7. Paddle Locator Test

The paddle linear path is divided in 40 grid locations. When the paddle is moved to either extreme of the track, the reported paddle location is as follows:

To do the paddle location test:

☐ Power up the cameras and motors

☐ Run the **testloc -a** and **testloc -b** commands to test location tracking for paddle A and B respectively. In this test, the paddle will move to the left until it reaches the limit switch and

pause for 5 seconds; then it will move to the right until they hit the opposite limit switch and stop. During this process the program will output to the console the grid location reported by the PaddleTracker thread.

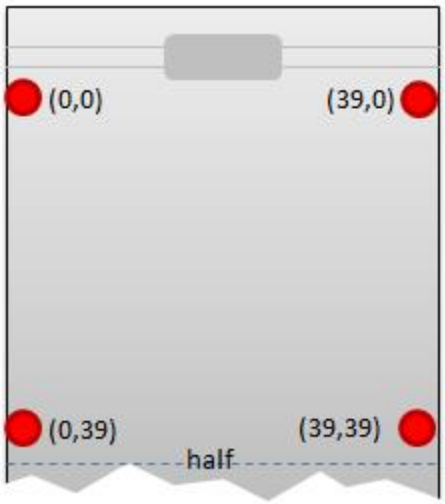
- ☐ Verify that at the left-most paddle position the grid location displayed in the console is 0.
- ☐ Verify that at the right most paddle position the grid location displayed is 39.

If the values are not between 0 and 39 (inclusive) the cause could be:

- Camera is not aligned properly.
- Green filter is failing: the green color washed out or green saturation is not enough; check lighting conditions.

8. Centroid Test

A half-section of the table is divided in 40x40 grid locations. Placing the puck at each corner of this area should produce a (x,y) centroid output as shown below:



To do the centroid test:

- ☐ Power up the cameras
- ☐ Run the **testcen -a** and **testcen -b** commands to test centroid tracking for the puck A and B respectively. This command will continuously output to the console the (x,y) grid location reported by the Centroid thread.
- ☐ Manually, place the puck in each corner of the grid area as shown in the illustration above.

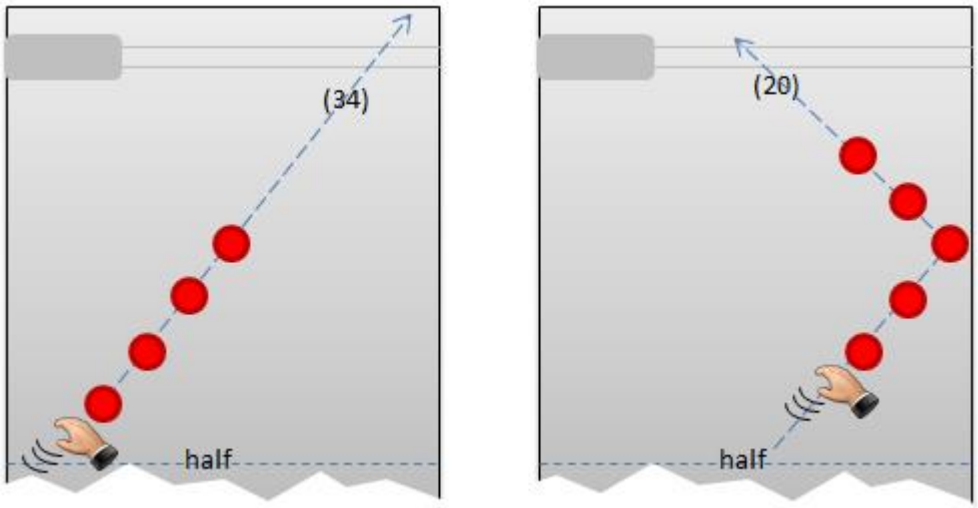
☐ Verify that for each corner, the correct (x,y) values are reported

If the (x,y) values are not between 0 and 39 (inclusive) the cause could be:

- Camera is not aligned properly.
- Red filter is failing: the red color washed out or red saturation is not enough; check lighting conditions.

9. Trajectory Test

This test has to be carefully executed by manually throwing the puck to a known grid location in the paddle's path. The throw can be a *direct shot* or *a rebound* off the side wall to test the reflective algorithm. This is shown in the following illustration:



To do the trajectory test

☐ Power up the cameras

☐ Run the **testtra -a** and **testtra -b** commands to test trajectory tracking for the table half A and B respectively. This command will continuously output to the shell console the predicted grid location reported by the Trajectory thread where the puck shall intersect the paddle plane.

☐ Manually throw the puck to a consistent target under the paddle's track.

☐ Verify that for each throw the Trajectory thread reports values from 0 to 39.

☐ You may adjust the speed of the throw, faster and faster until the algorithm fails to track the puck accurately. This test is useful to explore the limits of the system.

List of Command Line Test Programs

Unit Test Command	Description
v4l2grab	Command line utility for grab a JPEG file from the desired video device
grab0 and **grab1**	Utility scripts used to grab a series of continuous frames and save them as *.ppm files: **grab0** grabs frames from /dev/video0, **grab1** grabs frames from /dev/video1. When run simultaneously they validate the *uvcvideo* driver can handle simultaneous USB streams
testlimit	Command to verify all limit switches are responding
testmotor	Command used to test paddle movement (stepper motor is operational), Depends on **testlimit**.
testlaser	Command used to verify the laser and the receptor sensor are working
testkicker	Command used to verify the solenoids are working
testloc	Command used to test and calibrate tracking of paddle location
testcen	Command used to test and calibrate grid region for centroid tracking of the puck
testtra	Command used to test and calibrate trajectory prediction of the puck