Computer Vision

In FRC Robot Designs

Table of Contents

Specifications

Goals

Develop a software/hardware sub-system that can be used for computer vision based auto-targeting and FRC field element recognition

Requirements

1. Solution needs to operate in real time on real hardware

2. Solution needs to be supported in simulation as well as real hardware
3. Solution must at least offer basic image processing libraries
4. Solution needs to be low cost

Desirable Features

1. Can segment objects (“particles”,”ROIs”) from images (using filters, moment analysis etc.)
2. Can identify specific objects in images (e.g. target tape pattern, totes etc.)
3. Object identification accuracy can be improved by training
4. Offers Multi-Processor Support (multi-core cpus, GPUs)

Available Software packages and APIs

1. CUDA (Compute Unified Device Architecture**)**
    - C++ API that includes compiler directives that support multi-processing
    - Only supported on NVIDIA hardware (graphics boards)
2. cuDNN (NVIDIA Deep Neural Network Library)
    - Provides high level API for “deep learning” programming
    - Uses CUDA compute engine as back end
    - Only supported on NVIDIA hardware
3. Caffe (Deep learning Framework)
    - Developed by the Berkley Learning and Vision Center
    - Provides Python or Matlab API shells to CUDA(GPU) or OpenCL(CPU only) compute engines
    - Caffe-cuDNN requires top-end NVIDIA hardware for training
    - Trained networks can be deployed on lower-end NVIDIA hardware (e.g. Jetson)
4. Digits (graphical interface for deep learning)
    - Requires top-end NVIDIA hardware and software
    - Trained networks can be deployed on lower level hardware (e.g. Jetson)
    - Supports “deep learning” algorithms (Neural networks)
5. OpenCL (Open Computing Language)

- Open source alternative to CUDA
- Runs on most hardware platforms (AMD,NVIDIA,INTEL)

6. OpenCV (Open Computer Vision)
    - Runs on most os platforms (Linux, OSX, Windows, Arm)
    - Supports image filtering
    - May also support object identification and training (need to investigate)
7. IMAQ (ni-vision)
    - National Instruments proprietary software (commercial)
    - Only supported on RoboRio hardware
    - No possibility of simulation support
    - Supports object segmentation only (no object classification or training)
8. GRIP
    - Recently adopted by First as an alternate image processing package (over ni-vision)
    - Uses OpenCV image processing background engine
    - Uses JavaFX for front end UI
    - Can be built from source on Windows and Linux (Mac?)
    - Has GUI tuning application and headless implementation for deployment

Target Hardware platforms

1. Multicore x86_64 CPU
2. Raspberry Pi
3. Jetson
4. RoboRio

Strategies

Image processing Strategies

Use a relatively simple image processing API (e.g. GRIP, IMAQ)

- The image processing system uses filters, color match algorithms etc. to isolate portions of a

camera image that meet certain criteria for special target objects (reflective tape patterns etc.)

- The image processing system performs an analysis on these regions of interest (ROIs) and extracts basic information such as length,width (in pixels) position of object center on the screen etc.
- The reduced ROI data is sent from the image processing system to the Robot application
- The Robot application uses the ROIs center point width, height etc. along with prior knowledge of the objects size and shape to determine both the distance of the object from the camera and it's horizontal and vertical offset angles
- The Robot application uses the geometric data obtained for the target to adjust it's heading, shooter angle etc. as appropriate

Use an advanced object recognition API such as cuDNN

- Similar to that described above except that a more advanced technique (e.g. cuDNN) is used to identify objects based on general training data rather than relying on a set of unique target properties that may not be available in all situations
- Using such advanced techniques might require a more capable compute resource such as a NVIDIA Jetson or graphics card

Deployment Strategies

Use a single processor for both image processing and Robot application

1. Scenario
    - Capture images on RoboRio and run separate processing application (or integrated library) to extract features
        - e.g. GRIP(application) or IMAQ(library)
        - Can use inter-process or network communication protocols if image processing is carried out in a separate application
    - Use extracted image features to direct control
        - eg. Change angle for shooter target
    - Send Images up to driver station for user feedback
        - target areas in images could be annotated (surrounded by boxes etc.)
2. Pros

- Simplest and cheapest to implement (since only one processor board is required)
- Can use similar a strategy for simulation (since everything must run on the Linux host)

3. Cons
   - May run into processing bottleneck problems

Use a separate computer for image processing on deployed hardware

1. Scenario
   - Use RoboRio for Robot application
   - Use a second computer (e.g. Raspberry Pi or Jetson) for Image processing
     - Camera data only captured on processing board
   - Image processor extracts feature information and sends it to the RoboRio (e.g. via local ethernet)
   - Image processor sends raw or annotated images up to driver station
   - Roborio uses feature information to direct control
2. Pros
   - Uses an external processor to reduce computation overhead on RoboRio CPU
   - Has the potential to employ advanced object recognition algorithms such as NVIDIA cuDNN or openCV
3. Cons
   - Adds expense and complexity

Process raw images on host platform (driver station)

1. Scenario
   - Use RoboRio for Robot application
   - Send raw images from camera to driver station
   - Process images on driver station and extract features
   - Send feature information back to RoboRio to direct control
2. Pros
   - Can use high performance laptop to do the image processing

- In teleop mode the driver probably wants to see the camera images anyway
- No extra hardware to buy

3. Cons
   - May suffer from network IO bottlenecks because of the need to transfer images from the RoboRio to the Driver Station laptop
     - In competition, radio bandwidth is limited (e.g. 7MB/s) which will reduce image frame rate for both teleop and autonomous modes
   - Requires a higher performance driver station computer
   - Since the host data station does not run a RTOS (it runs Windows 10) image processing latencies will be indeterminate

Simulation Strategies

At the present time all simulation processing needs to run on the same (multi-core) CPU in a Ubuntu 14.04 operating system environment. In addition, because of a bug in Gazebo that causes a loss of depth information in the (simulated) camera images, the computer needs to run Linux natively (e.g. in a separate boot partition) and be equipped with a middle to high end graphics card (e.g. an integrated Intel GPU running Linux also exhibits the problem). The depth ordering bug is also evident when running Gazebo in a Virtual machine (under Windows) even on a system with a high end NVIDIA graphics card

Requirements

From a software point of view the vision interface to the Robot application should not require any (or much) special coding in order to run in the simulator or on real hardware

Strategy

- Capture a (simulated) camera publication from Gazebo and convert the image data to a form that can be input to an application like GRIP, raw openCV or an advanced API such as cuDNN
- Process the image using one of the tools mentioned above and extract relevant object position information such as center of mass, height, width etc.
- Pass the position information for the identified object (or objects) in a array of structures using a protocol that is also available on real hardware (e.g. the FRC NetworkTables protocol)
- Let the Robot application decide what to do with the object position data

Software SubSystems

GRIP

Notes for building and testing on Ubuntu 14.04

- note: GRIP can also be built on Windows (but this was not tested)

References

1. https://wpilib.screenstepslive.com/s/4485/m/50711/c/150895

   GRIP “Alpha”

2. https://usfirst.collab.net/sf/projects/grip_computer_vision_engine/ (?)

   note: this doesn't appear to be the same source tree that is described in the screenstepslive web page above (orphaned project ?)

3. https://github.com/WPIRoboticsProjects/GRIP/wiki/Setting-up-build-tools

Installation

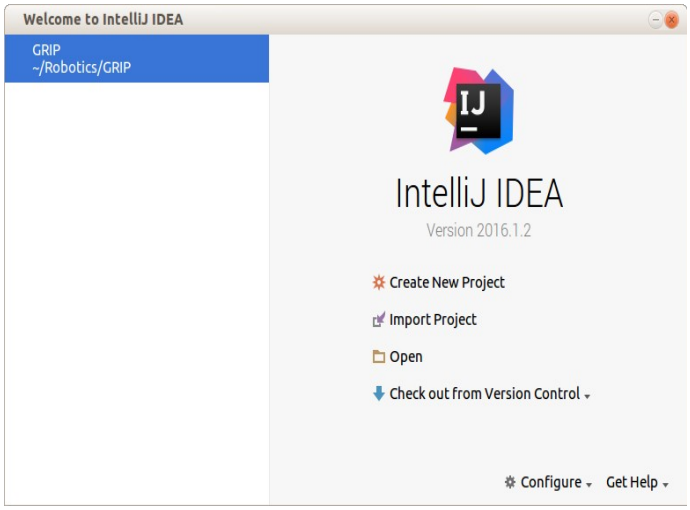
1. Clone the git repository

   git clone https://github.com/WPIRoboticsProjects/GRIP

   - creates a subdirectory GRIP in parent directory (i.e. where git clone was issued from)

2. Install IntelliJ (recommended Java IDE from ref. 3 above)
   - download from: https://www.jetbrains.com/idea/#chooseYourEdition (Community version)
     - ideaIC-2016.1.2b.tar.gz
     - installation instructions: https://www.jetbrains.com/help/idea/2016.1/installing-and-launching.html?origin=old_help#d1799863e158
   - Untar .gz file into public directory (e.g. /opt)
     - created directory: /opt/idea-IC-145.972.3
   - run installation script
     - cd /opt/idea-IC-145.972.3/bin
     - ./idea.sh
       - prompts for root password, then installs launcher script: /usr/local/bin/idea
   - Open GRIP project from IntelliJ

- $ idea
- Choose Import project



- Browse to GRIP main directory and select build.gradle as project file

Building and Launching

1. Build UI and Run from command line (instructions from README.md)

   $ ./gradlew :ui:run

2. Build and run UI from IntelliJ
   - (menu)Run->Edit Configurations
   - press “+” and then select new Gradle configuration
   - Enter a configuration name (e.g. “Run UI”)
   - press “box” icon to right of “gradle project” entry line
   - Select GRIP:ui from popup menu
   - enter “:ui:run” on “tasks” line
   - press “OK”
   - (menu)Run ..
   - enter “Run UI” from popup menu

Investigation Plan

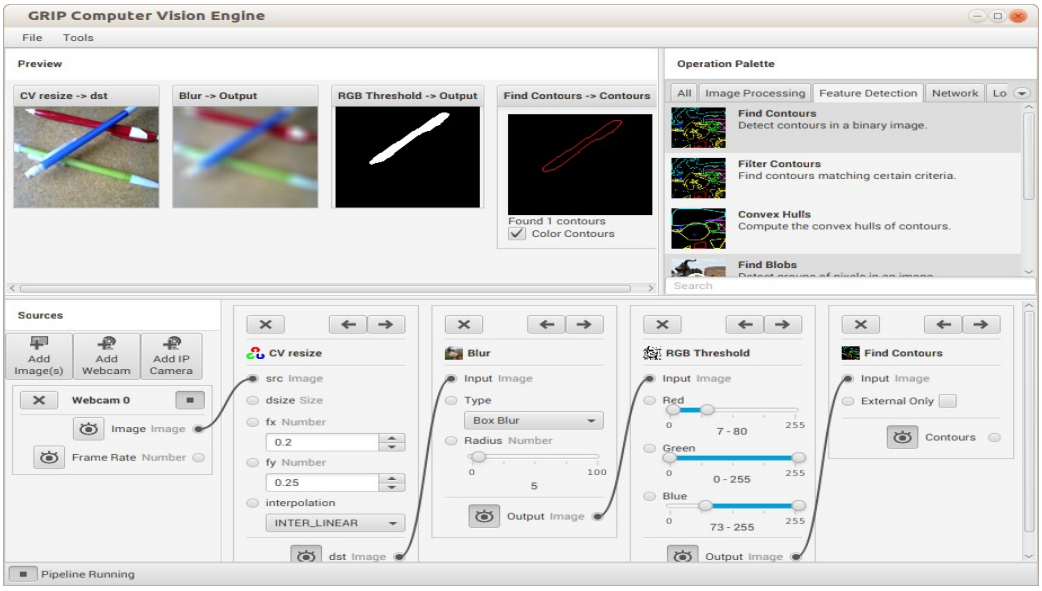
1. Build and Run JavaFX GRIP GUI on Ubuntu
2. Test Headless GRIP interface

- Use print statements, write to log file or look at network tables GUI to validate

3. Develop interface between Gazebo output (Camera publication) and GRIP input (UI and headless)
   - Emulate webcam or mpeg stream input modes ?
4. Test or develop interface between GRIP output and Eclipse Robot application (FRCUserProgram)
   - In simulation mode everything runs on the CPU
   - In deploy mode FRCUserProgram runs on the RoboRIO (but will not test this)
   - Q: Can FRCUserProgram intercept networkTables publications ?
5. Test or develop interface between GRIP output and Smart Dashboard
   - Smart Dashboard normally intercepts networkTables publications from FRCUserProgram
     - Q: Can it also capture networkTables publications from GRIP ?

Tests

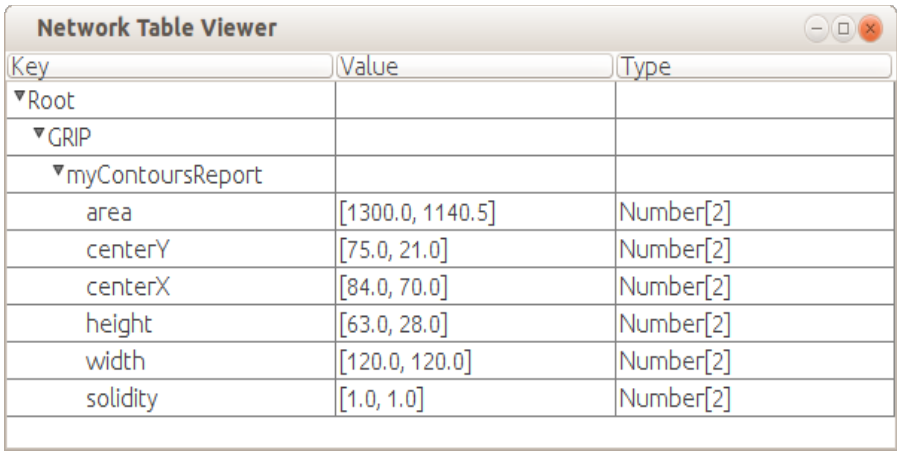
1. Find the Blue pencil using GRIP UI panel



2. Test Network tables publication
   - Add NTPublish CountoursReport panel and connect to “FindContours” panel

- Run “OutlineViewer”
  - $ java -jar ~/wpilib/tools/OutlineViewer.jar
  - Start “Server” (leave entry fields blank)

    note: If Server not started get “Connection refused error” in GRIP.log



Key	Value	Type
▼Root		
▼GRIP		
▼myContoursReport		
area	[1300.0, 1140.5]	Number[2]
centerY	[75.0, 21.0]	Number[2]
centerX	[84.0, 70.0]	Number[2]
height	[63.0, 28.0]	Number[2]
width	[120.0, 120.0]	Number[2]
solidity	[1.0, 1.0]	Number[2]

3. Capture GRIP NetworkTables publications in a Robot Program (linux_simulate build)
   - Run the GRIP “Find blue pencil” project with the NTPublishContoursReport panel connected to the “FindContours” panel as described above
     - make sure “Publish area” is checked in the NTPublishContoursReport panel
   - In Eclipse, write the following small “SampleRobot” program (e.g. in a FRC c++ project called GripTest):

```
class Robot: public SampleRobot {

public:
    std::shared_ptr<NetworkTable> table;
    Robot() {
        table = NetworkTable::GetTable("GRIP/myContoursReport");
    }
    void RobotInit() {
        static unsigned int old_area = 0;
        while (true) {
            std::vector<double> arr = table->GetNumberArray("area",
                    llvm::ArrayRef<double>());
            if (arr.size() > 0) {
                unsigned int new_area = (unsigned int) arr[0];
```

```
                if (new_area != old_area) {
                    std::cout << "area=" << new_area << std::endl;
                    old_area = new_area;
                }
            }
        }
    }
    void Autonomous() {
    }
    void OperatorControl() {
    }
    void Test() {
    }
};
START_ROBOT_CLASS(Robot)
```

- Compile “GripTest” using linux_simulate build
  - Generates FRCUserProgram in project's linux_simulate sub-directory

- In a command shell, start up gazebo
  - $ gazebo
  - note: not actually using gazebo in this case but FRCUserprogram wont start unless Gazebo is running
- In a command shell start up “FRCUserProgram”
  - cd GripTest/linux_simulate
  - $ ./FRCUserProgram
  - Example output in the command shell:

….

area=1000

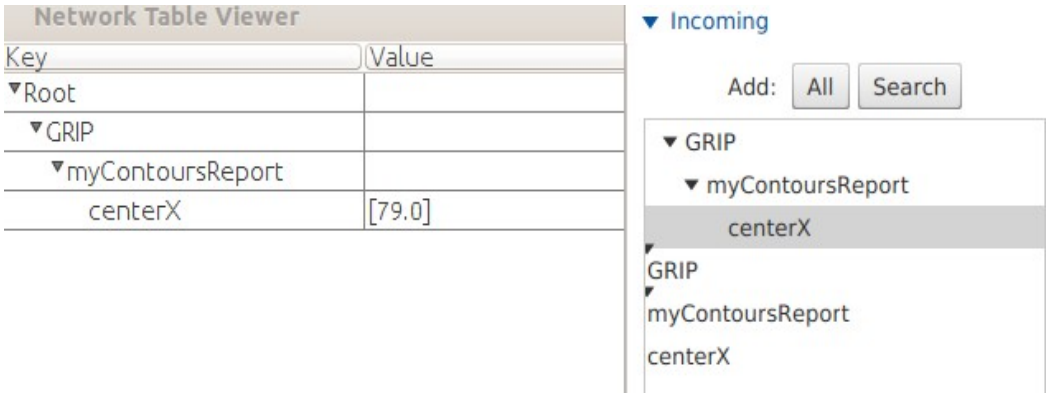
area=998

area=778

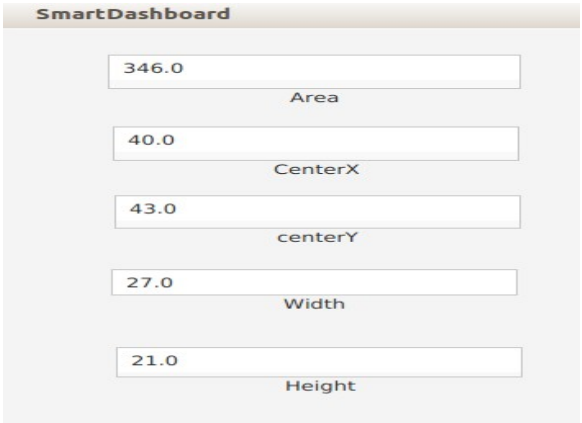
area=967

….

4. Show data from GRIP ContoursReport in SmartDashboard
   - Start SmartDashboard
   - Open Incoming list



- Drag an item (e.g. centerX) from “incoming” tree into panel (fails)
    - A Popup appears that offers a list of widget choices (e.g. number label,value meter etc.) but nothing gets generated in the panel
- Add an item manually (works)
    - open “+” ->Toolbox-general list and select “Array View” widget and drag to panel
    - double click array widget in panel and set Path to /GRIP/myContoursReport/centerX (etc.)
    - press labeler button and set label
        - “left” orientation doesn't resize column enough to show text but “bottom” seems to work (see below)



5. Build Smart Dashboard from source code
    - Was able to checkout the SFX project(s) from wpi git repository:

      https://usfirst.collab.net/sf/scm/do/listRepositories/projects.smartdashboard2/scm

but haven't yet been able to build it (note: needed to also get Netbeans 8.1 etc.)

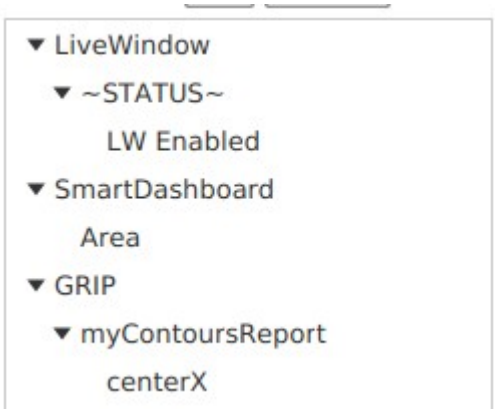
There seems to be some missing or changed URLs used in the build scripts (e.g. build couldn't find netbeans plugins at ..usfirst.collab.net:29418/netbeans_plugins -> repository doesn't exist)

6. Show GRIP data routed through FRCUserProgram in SmartDashboard

- Added the following line to the sample robot program shown above that captures GRIP data:

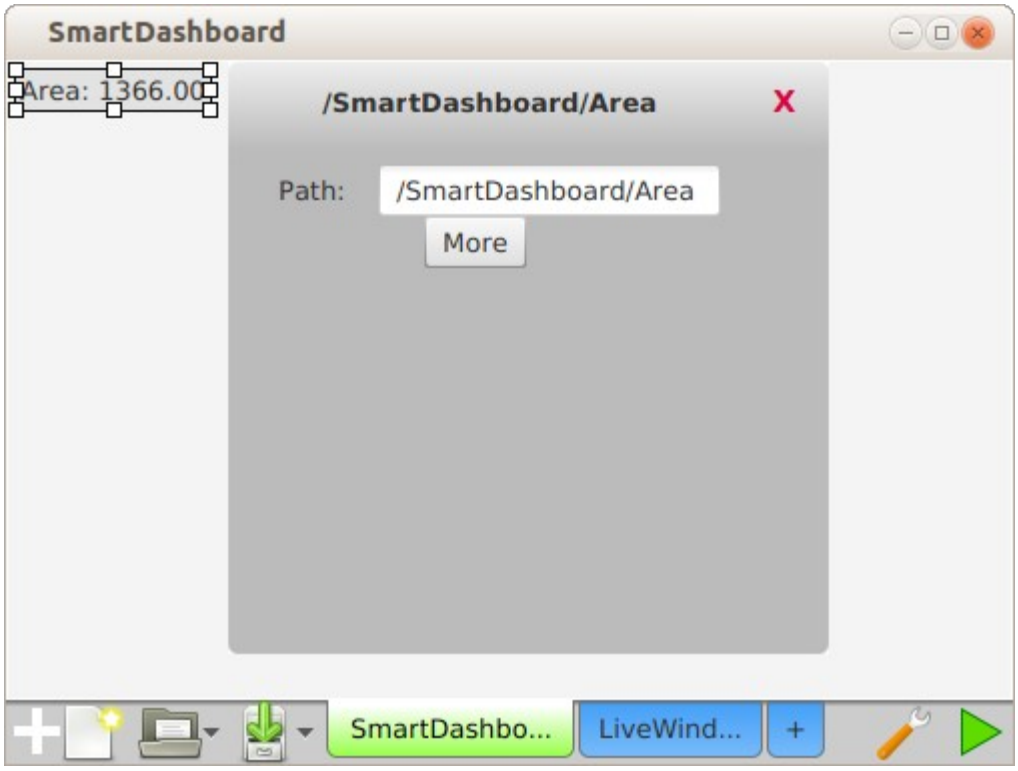
```
if (new_area != old_area) {
     std::cout << "area=" << new_area << std::endl;
  →  SmartDashboard::PutNumber("Area",new_area);
     old_area = new_area;
}
```

- built and started FRCUserProgram (and gazebo)
- Started SFX and opened “incoming” tab (now see SmartDashboard node in tree)

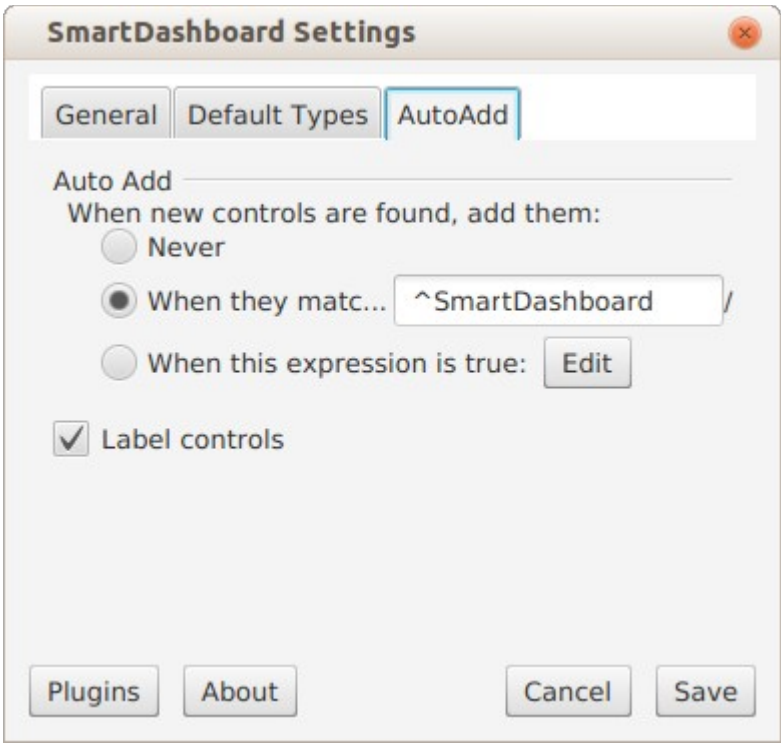


- Added a “number” label widget to canvas and set Path to /SmartDashboard/Area

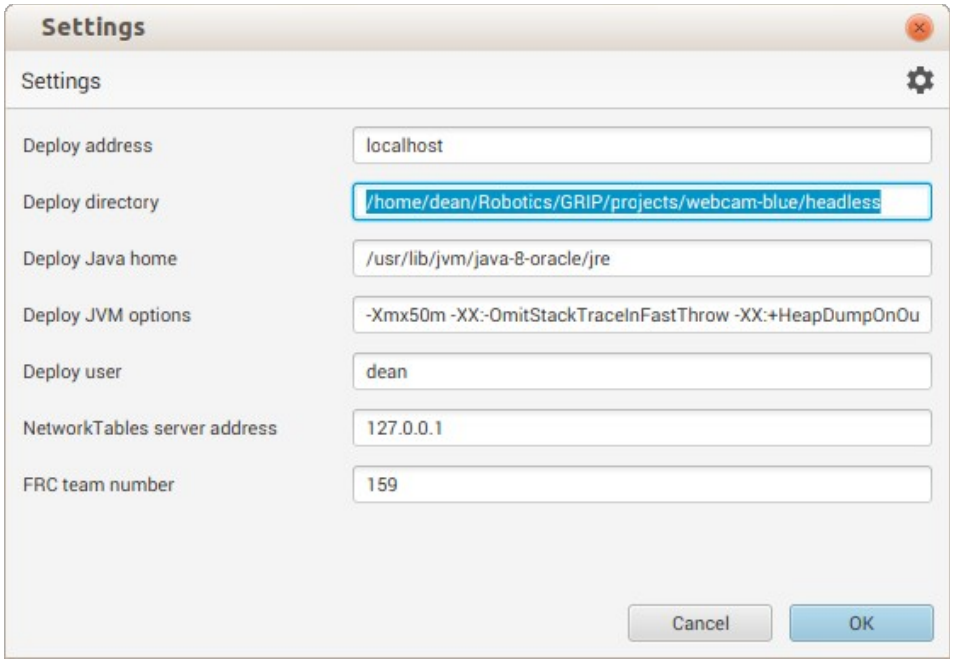
- GRIP “Area” numbers now get updated in data field



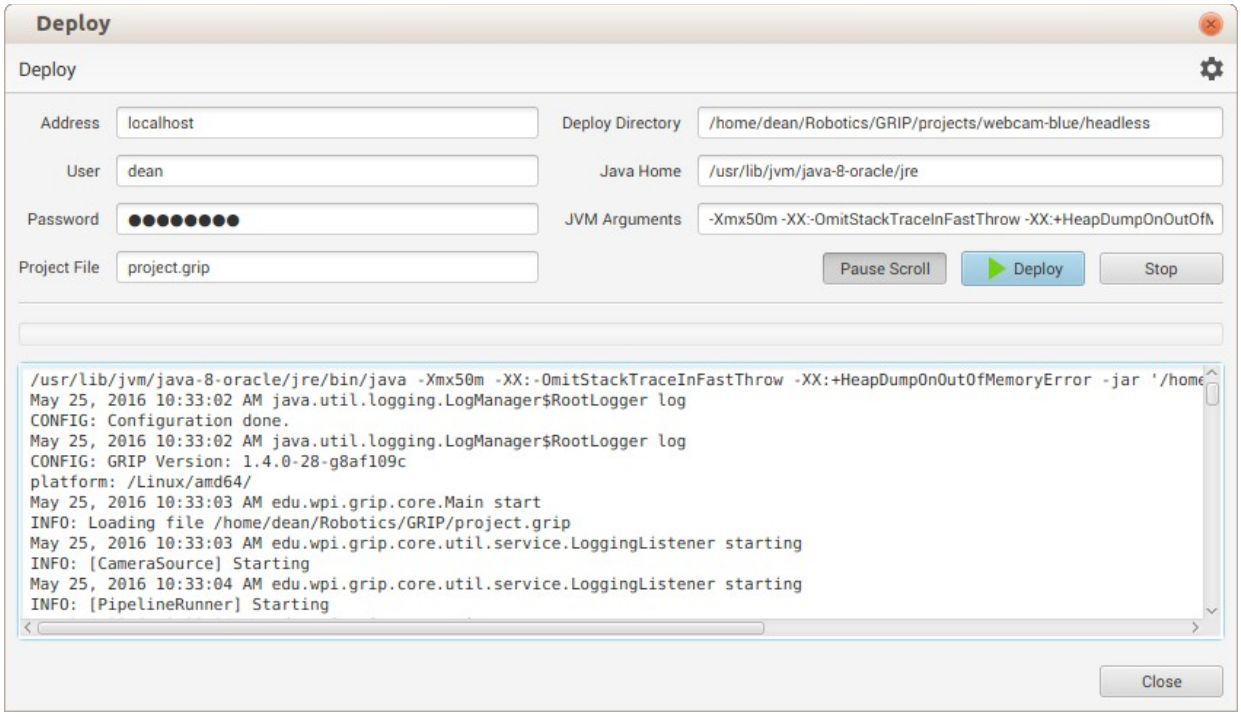
- Can also cause “Area” widget to be automatically created on startup by changing AutoAdd settings as shown:



7. Run GRIP ui from command line (i.e. not using gradlew)
    - $ gradlew :ui:installDist
        - creates startup shell script “ui” in GRIP/ui/build/install/ui/bin
    - $ ui/build/install/ui/bin/ui
        - starts up GRIP UI
    - $ ui/build/install/ui/bin/ui <path-to-project-file>
        - starts up GRIP UI and opens project file (e.g. project.grip)
8. Setup and run “Headless” GRIP project from command line (on localhost)
    - Start up GRIP UI and load project (using any method described above)
    - Open <menu>Tools->“Settings” dialog



    - Press <menu>Tools->Deploy
        - Brings up Deploy dialog



    - Fill in password (for ssh access) and “Deploy Directory” field
    - Press the “Deploy” button
    - Creates a startup script (grip), a headless jar file (grip.jar) and project file (project.grip) in the directory specified by the “Deploy Directory” field
- Run headless grip with project file
    - $ cd projects/webcam-blue/headless/
    - $ grip
    - See same command line output as above (but no UI)

Comments on GRIP

Pros

1. Intuitive user interface for testing the effect of common image processing operations
2. JavaFX UI Can be built and deployed on both Linux and Windows (Mac?)
3. Can run “headless” on Raspberry Pi, Roborio or Jetson (purportedly)
4. Supports ROS (gazebo) and networkTables (First/WPI) publication protocols

5. Can extract and publish geometry information from contour lists (center, width etc.)

Cons

1. Current implementation doesn't support generalized object recognition through “learning” algorithms
    - Uses basic image processing functions like “blur”, edge extraction etc.
    - Similar to ni-vision (IMAQ)
2. GRIP documentation says GPU isn't supported on Jetson board
3. On Raspberry Pi, problems with image publication require complicated workaround
4. Requires installing Java runtime on target platform
5. Java based (vs compiled binary), so processing speed may be an issue (?)

Problems

1. Can't figure out how to combine contours and original image
    - Want to see contours or convex hulls super-imposed on original image (e.g. for teleop mode)
    - Combining Original image with threshold output (as mask) is black everywhere except where mask is white
    - UI doesn't permit connection between original image and “contours” output as second image for any CV operations (bit-wise xor etc.)
    - No obvious tool to convert “Contours” or “Hulls” display back into RGB to make CV combine operators work
2. Image publication for display on Linux doesn't seem to work
    - May need GRIP code changes

Future Tasks

1. Interface GRIP input source to Gazebo camera messages (DONE)
2. build and deploy on Raspberry Pi (DONE)
3. build and deploy on Jetson (ARM processor only)
4. Modifications WPI GRIP source project
    - Add CV block Combine images (annotate images)

- Make URL source for mpeg entry block editable
- Add block to use object recognition data (advanced OpenCV feature)
- Modify Image Publish block to send output images to SmartDashboard or web browser on Linux

OpenCV

Version: OpenCV 3.1

target system: Ubuntu 14.04

references

http://www.askaswiss.com/2016/01/how-to-install-opencv-3-1-python-ubuntu-14-04.html

Building and Installation

1. Obtain source code via download
   - go to download site: http://opencv.org/downloads.htm
   - Select OpenCV for Linux/Mac
   - Download opencv-3.1.0.zip (e.g. to ~)
   - $ cd ~/
   - $ unzip opencv-3.1.0.zip
2. Build
   - $ cd opencv-3.1.0
   - $ mkdir build && cd build
   - $ cmake ..
   - make -j6
3. Install
   - $ sudo make install
   - $ sudo /bin/bash -c 'echo "/usr/local/lib" > /etc/ld.so.conf.d/opencv.conf'
   - $ sudo ldconfig

Alternate build using opencv git rep

1. Obtain active source code

- $ git clone https://github.com/Itseez/opencv
- cd opencv
- git checkout tags/3.1.0 -b 3.1.0
    - note: master branch is actually at 2.4 which produces compile errors unless CUDA is disabled

2. configure
    - mkdir build && cd build
    - ccmake ..
        - select options
            - YES: WITH_CUDA, WITH_OPENGL, WITH_MP ..
        - press c (configure)
        - press g (generate)
3. build
    - make -j6 (takes ~ 1hr to build)

Test Installation

1. Build samples
    - $ cd ../samples
    - mkdir test && cd test
    - $ ccmake ../
        - set OPENCV_DIR to ~/opencv/build
        - set OPENCV_FOUND to YES
    - $ make -j6
2. Test
    - fix data paths so that samples will work using defaults
        - cd ~/opencv
        - cp -R data/* samples/data (training data needed for face-detect)
        - $ cd cpp

- ln -s ~/opencv/samples/data ../data

- $ cpp-example-facedetect data/lena.jpg
  - should see woman's head with face and eyes circled

Mjpg Streamer

Mjpg_steamer is a utility application that can be used to convert a set of images or the output of a USB WebCam into a TCP/IP data stream that can be input to an external application, web browser or GRIP (as an “ip camera”). It is a github project that can be build from source on most hardware/software platforms (including the Raspberry Pi)

1. Installation (Linux)
   - Obtain libraries and source code
     - sudo apt-get install libjpeg8-dev
     - git clone https://github.com/jacksonliam/mjpg-streamer.git
   - Build from source and install
     - cd mjpg-streamer/mjpg-streamer-experimental
     - make clean all
   - Install
     - sudo make install
2. Options
   - Show help/usage
     - mjpg_streamer [-i, -o] '<plugin-name> --help'
       - mjpg_streamer -i 'input_file.so –help'
       - mjpg_streamer -o 'output_http.so --help'
   - -f <path>
     - set path for saved jpeg files
     - e.g. : -f /tmp/shooter-camera
   - -r
     - remove jpeg files after reading (otherwise keep all files)

3. Tests
    1. pipe some jpg files in a directory to a web address
        - mjpg_streamer -i "input_file.so -f /home/dean/test/images -r -d 0.1" -o "output_http.so -w www -p 1180"
        - In Web browser (e.g. Firefox) capture and display images or image stream
            - e.g. Set web address to http://Ubuntu14.local:1180/?action=stream

Gazebo Simulation with GRIP

Setup

1. Modify simulation FRC Robot sdf file

- `Add a “save” element in the sensor/camera block e.g:`

```
<sensor name='ShooterCamera' type='camera'>
      <visualize>1</visualize>
      <always_on>1</always_on>
      <update_rate>2</update_rate>
      <camera name='Shootercamera'>
            <save enabled="true">
            <path>/tmp/shooter-camera</path>
            </save>
      </camera>
</sensor>
```

- When the simulation is run this will generate a set of jpg files to appear in the “path” directory
- Use the “`update_rate” element in the sensor block to set the file capture frame rate`

Run Simulation

2. start mjpg-streamer with arguments that will generate an mpeg stream using the files in the indicated directory
    - rm -fr /tmp/shooter-camera
    - mkdir -p /tmp/shooter-camera

- mjpg_streamer -i "input_file.so -f /tmp/shooter-camera -r -d 0.1" -o "output_http.so -w www -p 1180"

3. Start GRIP as described above and set input source to a new ip camera with local URL

   e.g. http://Ubuntu14.local:1180/?action=stream

4. Start Gazebo Robot simulation as described previously (__ref__)
   - cd to FRC Robot project and run startup script e.g.:

     # start gazebo

     export SWMODEL=$SWEXPORTS/2016-Robot_Exported

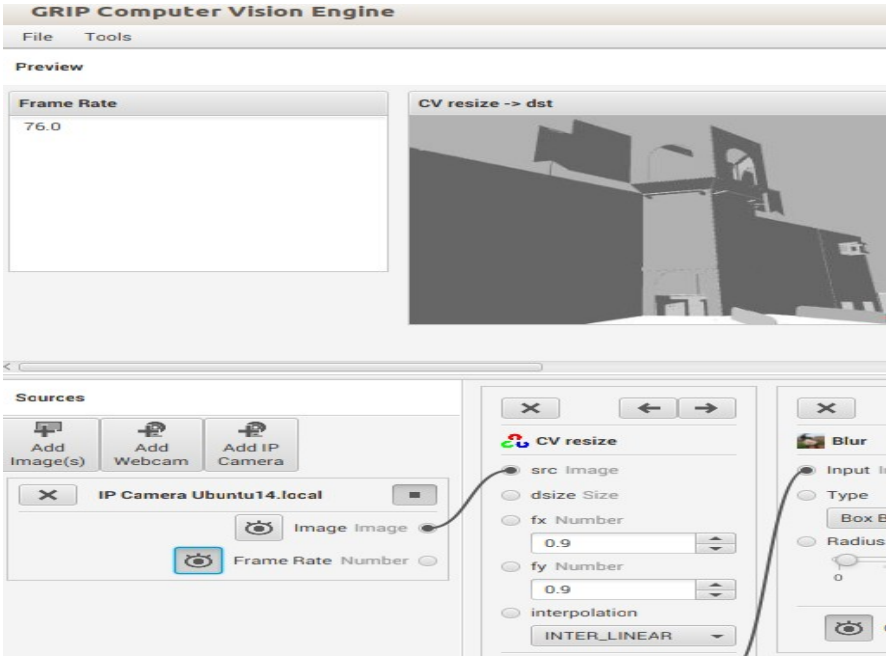
     export GAZEBO_MODEL_PATH=$SWMODEL:$WPILIB/simulation/models:/usr/share/gazebo-6.5/models

     gazebo --verbose $SWMODEL/2016-Robot-field.world

   - Connect Joystick or Gamepad and start sim_ds
     - $sim_ds
   - Start FRCUserProgram
     - $ cd <path-to-robot-project>/linux_simulate
     - $ ./FRCUserProgram

First Results

1. Screenshot of Gazebo simulation of shooter camera on team 159's 2016 robot in FRC Stronghold Challenge field



Improvements

Add colored tape strips around tower targets

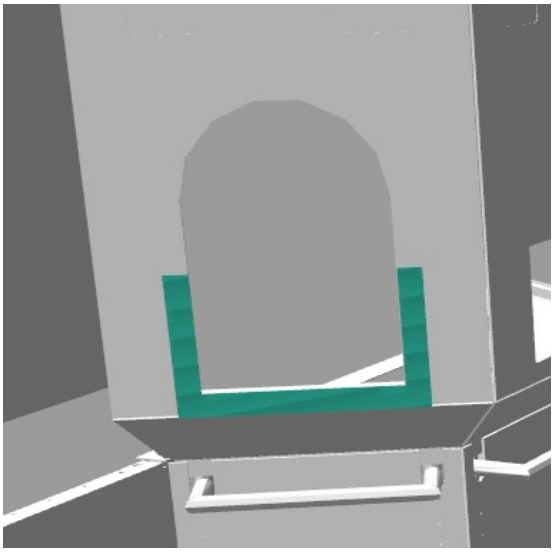
1. Create a tape pattern sdf file using solidworks exporter
    - create a sketch bottom = 18”, sides=12”, tape width=2”
    - Create a part then assembly from sketch
    - extrude sketch a slight amount (e.g. ~0.1”)
    - generate reference properties for origin, baseplane and axis
    - Use gazebo exporter tab “edit model” to set properties
        - Set name to TapePattern
        - set origin, baseplane and axis using reference objects created previously
        - Select extruded sketch as base joint
        - set color
    - Export model
2. Modify sdf file (hand edit)
    - remove collision and inertia elements and set gravity=0

    - otherwise object will drop to the ground when physics is enabled
- edit ambient and diffuse colors if needed
    - for test purposes first try setting color to pure green(rgb=0,1,0)

3. Add TapePattern object to Robot simulation world file
    - Start Gazebo with world file containing robot and FRC field
    - Stop physics simulation
    - Insert a “TapePattern” object into gazebo
    - Use Position and Rotate modes to move TapePattern object to correct position below window in Turret
    - Select the TapePattern item in the world tree and Copy down “pose” settings
    - Exit gazebo
    - Edit the robot world file to add a tape pattern object with the correct pose e.g.:

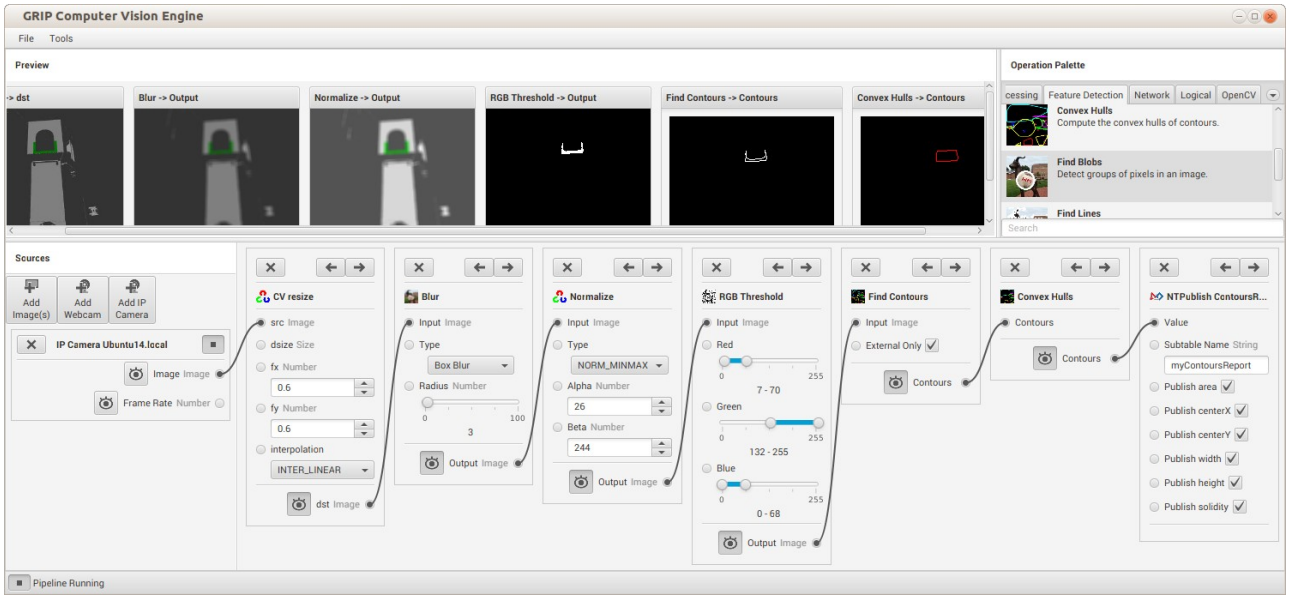
```
<include>
  <uri>model://TapePattern</uri>
  <pose>0.125288 -7.62138 2.11 0.000148 0.000257 0.52429</pose>
</include>
```

    - Screenshot of tape pattern in left turret



Optimize GRIP Processing blocks to identify tape pattern target

1. Start up mgpg-streamer (monitor image dump directory)
2. Start up Gazebo Robot-FRC field simulation
3. start up GRIP and set source to ip camera (use local host as URL)
4. In Gazebo teleop mode drive robot to location of tower and adjust shooter angle until target is visible in GRIP input window
5. Set processing blocks in GRIP
    - IP Camera input source (receiving data from mjpg-streamer)
    - resize (optional)
    - blur (kernel size=3)
    - normalize (default settings)
    - RGB Threshold (optimized for pure green tape pattern)
    - find contours (external only)
    - convex hulls
    - NT Publish Contours

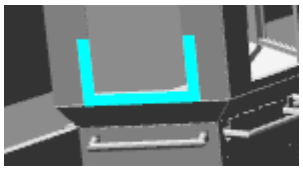


Optimize lighting to match FRC field environment

1. Is it possible to add a light source to the Robot ?

- No: not with latest version of Gazebo
- But is a feature request

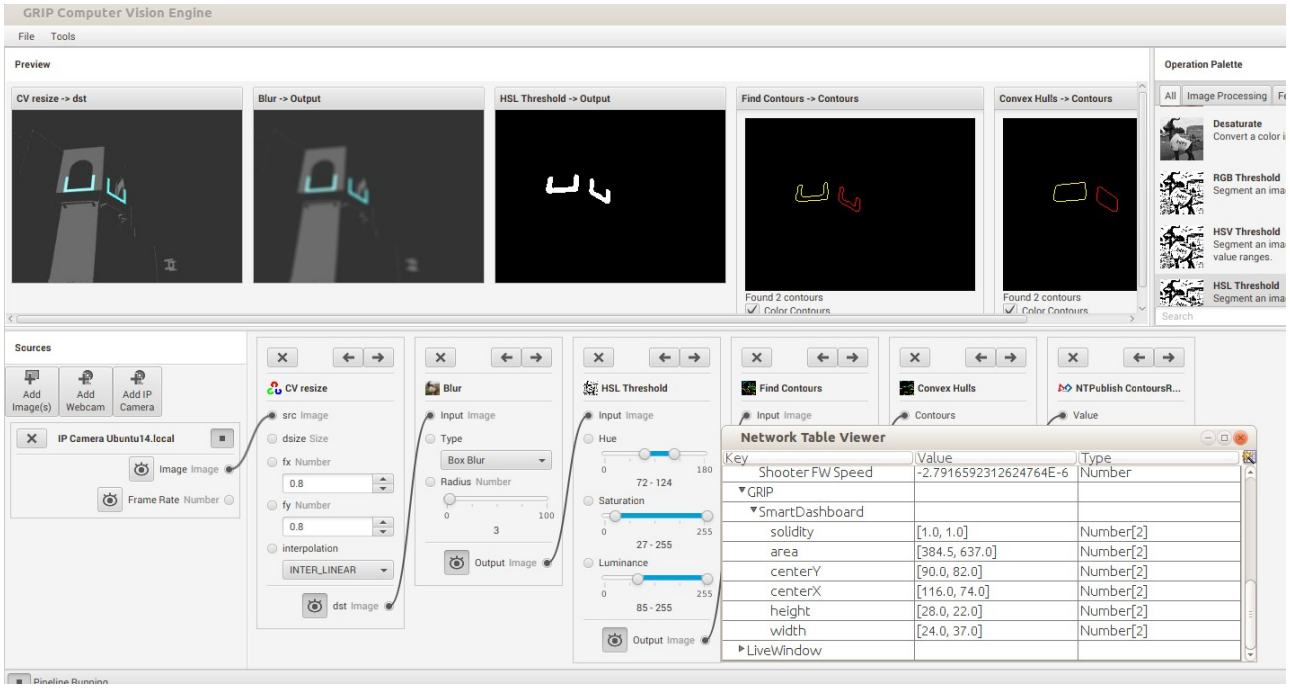
2. Is illumination supported as an sdf material property ?
   - Yes: set “emissive” property in material element
   - e.g. <emissive>0.0 1.0 1.0 1</emissive>



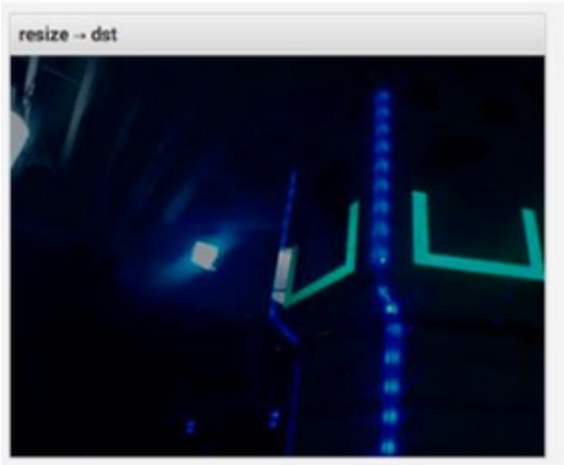
3. Determine best color for tape pattern
   - ambient and diffuse =1,1,1,1
   - <emissive>0.2 1.0 1.0 1</emissive>
4. Is it possible to modify the simulated camera properties such as “exposure” and “brightness” ?
   - Need to check
   - Can emulate effect by darkening scene lighting
5. Modify Gazebo ambient lighting properties be better emulate inside environment
   - Disabled global shadows
   - darkened field environment by changing rgb diffuse color values from 0.8 to 0.5

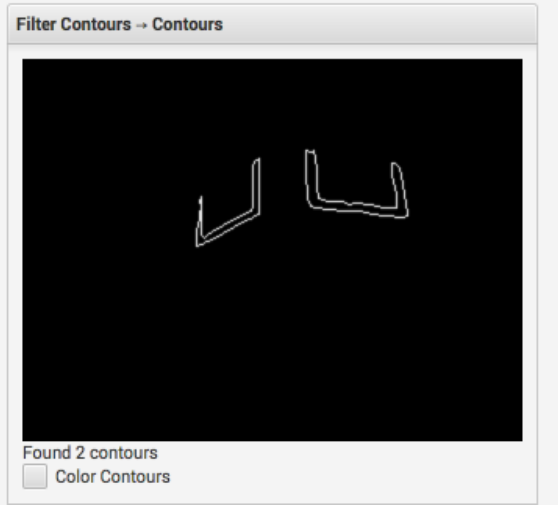
Improved Results

1. Added second tape pattern to center turret
2. Processing blocks
   - Got better contour isolation by using HSL vs RGB threshold block
   - Removed Normalize block
3. Screenshot of GRIP UI with improved target lighting



4. Comparison with real FRC field results
    - reference: http://wpilib.screenstepslive.com/s/4485/m/50711/l/481750-using-grip-for-the-2016-game





TODO

1. Modify Solidworks Gazebo exporter plugin to provide entry fields for jpeg output path to avoid having to hand edit sdf files

Jetson TK1

Setup

1. Hardware version
    - Tegra K1
    - price: $192, NVIDIA
2. Software
    - Linux tegra-ubuntu
    - kernel: 3.10.40 SMP PREEMPT
3. Cables etc.
    - AC Power Adapter
    - Ethernet cable (to switch)
    - USB to micro-USB cable
    - note: Jetson has built-in 16 GB flash memory

OS Installation

1. Setup
    - In order to view or download documentation you need to first become a member of the NVIDIA “Embedded Computing” group
    - Then go to Download site and download
        - “Jetson TK1 User guide”
        - “Jetpack for L4T (Ubuntu 64 bit)”
            - https://developer.nvidia.com/embedded/dlc/jetpack-l4t-2_2
            - Downloads: JetPack-L4T-2.2-linux-x64.run
2. References
    - Follow Instructions in TK1 user guide for flashing OS

Remote Desktop (VNC)

Setup VNC server and VirtualGL

1. References

- https://devtalk.nvidia.com/default/topic/828974/embedded-systems/-howto-install-virtualgl-and-turbovnc-to-jetson-tk1/

2. log onto Jetson
   - ssh -l ubuntu tegra-ubuntu.local
   - password is “ubuntu”
3. Install TurboVNV and VirtualGL on Jetson
   - log onto Jetson
   - Install pre-compiled ARM binaries given in reference

```
wget http://demotomohiro.github.io/hardware/jetson_tk1/pkg/libjpeg-turbo_1.4.2_armhf.deb
wget http://demotomohiro.github.io/hardware/jetson_tk1/pkg/virtualgl_2.4.1_armhf.deb
wget http://demotomohiro.github.io/hardware/jetson_tk1/pkg/turbovnc_2.0.1_armhf.deb

sudo dpkg -i libjpeg-turbo_1.4.2_armhf.deb
sudo dpkg -i virtualgl_2.4.1_armhf.deb
sudo dpkg -i turbovnc_2.0.1_armhf.deb
```

   - Add jpeg to library path

```
sudo vi /etc/ld.so.conf.d/libjpeg-turbo.conf
add the following line: /opt/libjpeg-turbo/lib32
```

   - Install xfce4 Desktop (optional)

```
sudo apt-get install xfce4 xfce4-goodies gnome-icon-theme-full
```

   - Configure vncserver (a dialog appears on first time launch of vncserver)
     - start vncserver
       - /opt/TurboVNC/bin/vncserver
       - Can answer “no” to questions about VNC user group etc.
       - Creates a ~/.vnc/xstartup.turbovnc file in home directory
     - stop (kill) vncserver
       - /opt/TurboVNC/bin/vncserver -kill :1
   - Set vnc preferences for remote display

```
vi .vnc/xstartup.turbovnc

add  lines:

unset SESSION_MANAGER
unset DBUS_SESSION_BUS_ADDRESS
```

```
startxfce4 &
```

- Modify xorg.conf (add a ”Screen” section)

```
$ sudo vi /etc/xorg.conf
```

- add:

```
Section "Screen"
   Identifier  "Screen0"
   Device "Tegra0"
   Monitor "DSI-0"
   Option "AllowEmptyInitialConfiguration"
   Option "UseEdid" "False"
EndSection
```

4. Configure Jetson to automatically start up a vncserver session (:1) on boot
   - `sudo vi /etc/sysconfig/tvncservers`
   - add the following 2 lines:

     `VNCSERVERS="1:ubuntu"`

     `VNCSERVERARGS[1]="-geometry 800x600” (optional)`

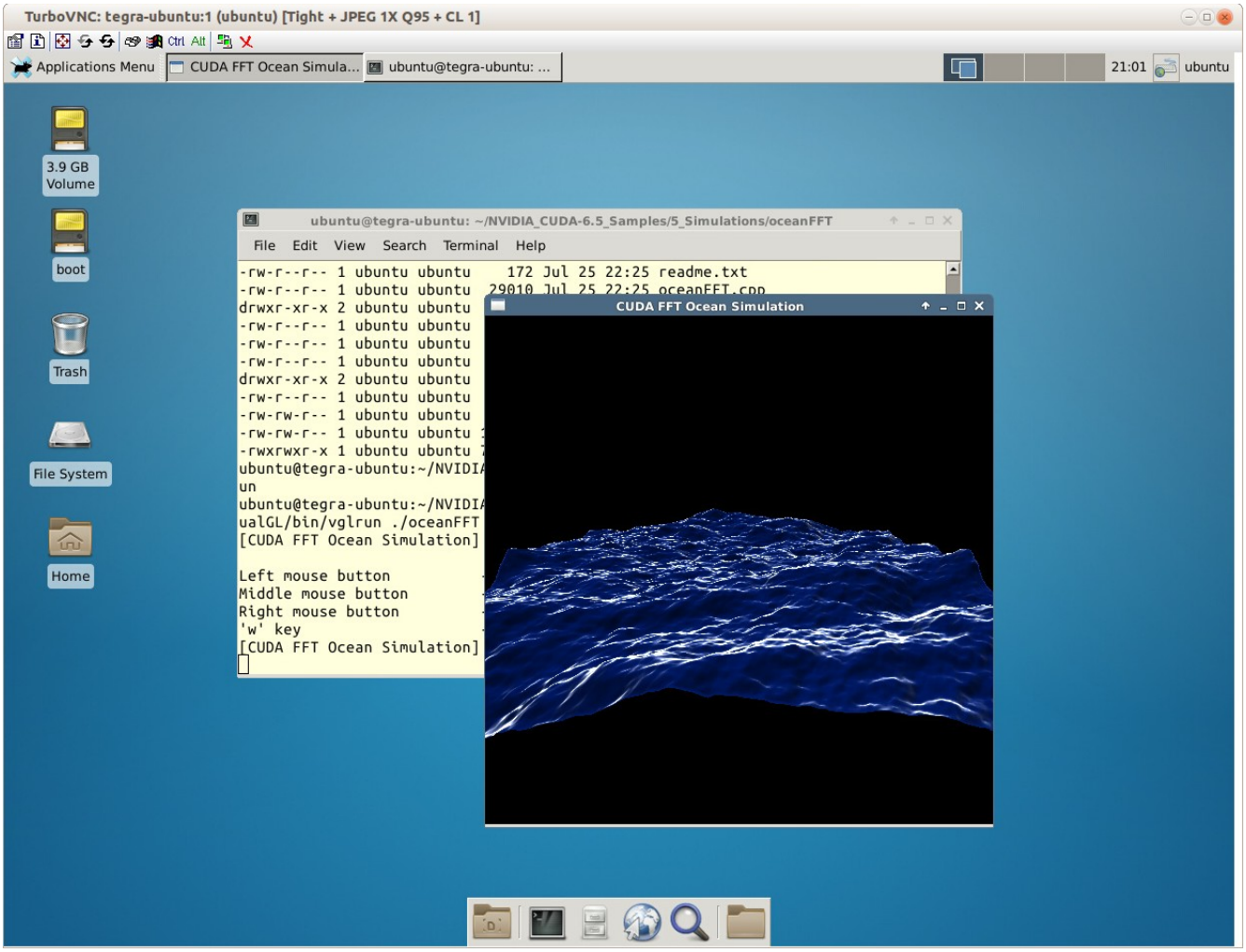
   - `sudo update-rc.d tvncserver defaults`
   - `To disable automatic startup on boot:`
     - `sudo update-rc.d tvncserver disable`
   - `To re-enable automatic startup on boot:`
     - `sudo update-rc.d tvncserver enable`

Test OpenGL Samples

1. Install TurboVNV [VirtualGL] on Ubuntu desktop
   - Download and install TurboVNC
   - Download and install VirtualGL (optional)
2. Compile NVIDIA samples on Jetson
   - Log on to Jetson from Ubuntu host
     - ssh -l ubuntu tegra-ubuntu.local
   - Build all samples
     - cd NVIDIA_CUDA-6.5_Samples
     - make -j4
3. Set VNC connection from Ubuntu desktop to Jetson

- On Jetson, vncserver should be running after a rebooting (see 3 above)
    - If not, start up a server session manually
    - /opt/TurboVNC/bin/vncserver :1
- On Ubuntu desktop connect to vncserver running on the Jetson
    - /opt/TurboVNC/bin/vncviewer tegra-ubuntu.local:1
    - Should see Jetson xfce4 desktop in new window

4. Running Samples
    - from vncviewer window of Jetson desktop on Ubuntu14 host open a terminal
        - note: originally the terminal was “black on black” so the cursor wasn't visible until I changed the color preferences
    - cd to Example directory
        - cd ./NVIDIA_CUDA-6.5_Samples/5_Simulations/oceanFFT
        - `/opt/VirtualGL/bin/vglrun ./oceanFFT`



5. Performance comparison (Jetson vs GTX 980 on Ubuntu host)
    - Sample : smokeParticles
    - FPS (GTX 980): 60 (pegged to max update rate)
        - NVIDIA “Samples” directory was version 7.5 (vs. 6.5 on Jetson)
    - FPS (Jetson) : 8
        - FPS increased to 16 when window size was maximally reduced

CUDA SDK

1. References

- https://petewarden.com/2014/10/25/how-to-run-the-caffe-deep-learning-vision-library-on-nvidias-jetson-mobile-gpu-board/

2. The reference describes how to download and install `cuda-repo-l4t-r19.2_6.0-`

`42_armhf.deb` but this was skipped because a later version (`cuda-repo-l4t-r21.3-6-5-local_6.5-50_armhf.deb`)is present (in ~/cuda-l4t) and seems to already be installed

3. Set PATH and LD_LIBRARY_PATH in ~/.profile
   - export PATH=`/usr/local/cuda/bin:$PATH`
   - export LD_LIBRARY_PATH=/usr/local/cuda/lib:$LD_LIBRARY_PATH
   - . ~/.profile
4. Verify that nvcc is installed
   - nvcc -V

     ```
     nvcc: NVIDIA (R) Cuda compiler driver
     Copyright (c) 2005-2014 NVIDIA Corporation
     Built on Tue_Feb_17_22:53:16_CST_2015
     Cuda compilation tools, release 6.5, V6.5.45
     ```

cuDNN

1. Installation
   - After the OS-Jetpack" installation there is a directory called cudnn in ~/ so it looks like cudnn (version 2) is already installed
2. Test
   - $ cd cudnn/cudnn-sample-v2
   - $ make
     - builds mnistCUDNN
   - $ ./mnistCUDNN

     Performing forward propagation ...

     Result of classification: 1 3 5

     ….

     Test passed!

OpenCV

1. Installation

- There is a directory called OpenCV4Tegra in ~/ so it looks like OpenCV is already installed

2. Verify installation
   - cd ~/OpenCV4Tegra
   - ./ocv.sh

     ….

     Reading state information... Done

     libopencv4tegra is already the newest version.

     libopencv4tegra-dev is already the newest version.

Caffe

Build

1. References
   1. https://petewarden.com/2014/10/25/how-to-run-the-caffe-deep-learning-vision-library-on-nvidias-jetson-mobile-gpu-board/
   2. http://jetsonhacks.com/2015/01/20/nvidia-jetson-tk1-cudnn-install-caffe-example/
   3. http://planspace.org/20150614-the_nvidia_jetson_tk1_with_caffe_on_mnist/
2. Install libraries

```
sudo apt-get install cmake libleveldb-dev libsnappy-dev
sudo apt-get install libprotobuf-dev protobuf-compiler
sudo apt-get install libboost-thread-dev libboost-system-dev
sudo apt-get install libatlas-base-dev libhdf5-serial-dev libgflags-dev
sudo apt-get install libgoogle-glog-dev liblmdb-dev
```

3. Install git
   - `sudo apt-get install -y git`
4. Clone caffe git repo
   - `git clone https://github.com/BVLC/caffe.git`
   - reference says to check out the “dev” branch but that seems to no longer exist
5. Build caffe (master branch)
   - following reference, installed g++ 4.7 to workaround some reported build problems with 4.8

        - sudo apt-get install gcc-4.7 g++-4.7
    - `customize Makefile`
        - `cd caffe`
        - `cp Makefile.config.example Makefile.config`
        - `sed -i "s/# CUSTOM_CXX := g++/CUSTOM_CXX := g++-4.7/" Makefile.config`
        - `vi Makefile.config`
        - `Uncommented “USE_CUDNN := 1”`
    - `make -j4`
        - `get compile errors: CUDNN_PROPAGATE_NAN undeclared etc.`
    - Did a web search and it looks like the master BVLC branch of caffe isn't compatible with cudnn v2 (which is what gets installed on TK1 by Jetpack 2.2)
        - tried checking out tag “rc2” from Berkley caffe site but that also failed to compile
        - Also got build problems when building from NVIDIA github site: https://github.com/NVIDIA/caffe
        - finally found an alternate site which is compatible with cudnn v2:
            - git clone https://github.com/slayton58/caffe
    - Rebuilt from alternate repo with success
        - `make -j4`
        - note: Also didn't need to modify `Makefile.config to use g++ 4.7`

Test

1. Build and run “runtest”
    - make -j4 runtest
    - passed many tests but ultimately failed with: MDB_MAP_FULL error
    - found a possible workaround on web at: http://jetsonhacks.com/2015/01/17/nvidia-jetson-tk1-caffe-deep-learning-framework/
        - Change ../examples/mnist/convert_mnist_data.cpp:89:56
        - CHECK_EQ(mdb_env_set_mapsize(mdb_env,), MDB_SUCCESS)

- to: CHECK_EQ(mdb_env_set_mapsize(mdb_env, 536870912), MDB_SUCCESS)

- rebuild and run examples
  - make

    CXX examples/mnist/convert_mnist_data.cpp

    CXX/LD -o .build_release/examples/mnist/convert_mnist_data.bin
  - make -j4 runtest
    - still fails (same error)
  - try a clean caffe build
    - make clean;
    - make -j4 all
      - got number overflow warning for: found another src/caffe/util/db_lmdb.cpp
      - again changed value from 1099511627776 to 536870912
    - make -j4 all
  - make -j4 runtest
    - probably ok now (had to leave after watching it run for over an hour and on return was loggod out from Jetson – maybe because screen saver kicked in at some point)

2. Performance tests (from reference 1)

- syntax: build/tools/caffe time -model=models/bvlc_alexnet/deploy.prototxt [-gpu=0]
  - Syntax reported in reference (--gpu, --model) didn't work (need to use -model, -gpu)
  - add -gpu=0 to run with GPU support (else uses CPU)
- Jetson TK1 (USE_CUDNN := 1)
  - GPU
    - Often seems to hang on “Performing Forward” (no output to screen)
    - Average Forward-Backward: 525.371 ms.
    - Value reported in reference 1 : 337.86
    - Value reported in reference 2 : 517.052 ms
  - CPU

        - Average Forward-Backward: 10764.6 ms.
        - Value reported in reference 1: 585 ms
        - note: Observed “Average” result significantly (x20) worse than that reported in the references
    - GPU/CPU speedup = 20.5
- Jetson TK1 (# USE_CUDNN := 1)
    - Average Forward-Backward:
        - GPU: 513.565 ms
        - CPU: 10754 ms
- Jetson TK1 (# USE_CUDNN := 1 CPU_ONLY := 1)
    - Forward-Backward: 10896 ms
- GTX 980 GPU on AMD Phenom-II CPU
    - cudnn 5.1, CUDA 7.5, caffe (latest master branch)
    - GPU: 16.8638 ms.
    - CPU: 3320.1 ms
    - GPU/CPU speedup = 207
- Comments
    - No GPU speedup observed with cuDNN (v2) enabled
    - GPU performance comparable (but slightly worse) to that reported elsewhere
    - CPU performance **20x** worse than that reported elsewhere
        - not much change when configuring cpu for “max performance” e.g. as described in: http://elinux.org/Jetson/Performance

Raspberry Pi-3

Setup

1. Hardware version
    - Raspberry Pi 3 (model B)

- purchased from NewEgg ($35)

2. Software version
   - Raspbian Jessie-Lite
3. Cables & other hardware
   - micro SD Card reader
   - micro SD card
     - started with a 4G card but ran out of room when installing OpenCV so got a 16 G card
     - recommend 16G or greater
   - USB –> micro USB cable
   - Standard Ethernet cable

Installation of Operating system (from Ubuntu host)

1. Download Raspbian Jessie Lite OS
   - source: https://www.raspberrypi.org/downloads/raspbian/
   - download file: 2016-05-27-raspbian-jessie-lite.zip
   - $ unzip 2016-05-27-raspbian-jessie-lite.zip
   - creates: 2016-05-27-raspbian-jessie-lite.img (size:1387266048)
2. Install OS on SD card
   - Instructions: https://www.raspberrypi.org/documentation/installation/installing-images/linux.md
   - Determine the partition number for the SD card reader
     - Remove the SD card reader
     - $ df -h (note: the base configuration info)
     - Insert the SD Card reader with micro Sd card installed
     - $ df -h
     - These are the new devices which appear over the base configuration

       /dev/sdc1   818M 5.7M 812M   1% /media/dean/D8DE-D3EE

       /dev/sdc2   976M 135M 791M 15% /media/dean/ecabb50d-36f7-4ec3-b449-

11c76e8467f5

- The full partition for the card is therefore “sdc” (following the instructions in the ref)
  - note: when this procedure was repeated using a 16G card the partition was “sdg” instead

3. Burn the image onto the SD card
   - $ umount /dev/sdc1
   - $ umount /dev/sdc2 (if present)
   - $ sudo dd bs=4M if=2016-05-27-raspbian-jessie-lite.img of=/dev/sdc
   - $ sync
4. Check validity of image on card (optional)
   - $ sudo dd bs=4M if=/dev/sdc of=from-sd-card.img
     - ls -l: -rw-r--r-- 1 root root 3980394496 Jun 3 09:35 from-sd-card.img
   - $ sudo truncate --reference 2016-05-27-raspbian-jessie-lite.img from-sd-card.img
     - ls -l: -rw-r--r-- 1 root root 1387266048 Jun 3 09:37 from-sd-card.img
   - $ diff -s 2016-05-27-raspbian-jessie-lite.img from-sd-card.img
     - output: Files 2016-05-27-raspbian-jessie-lite.img and from-sd-card.img are identical
5. Connect up the raspberry Pi
   - Insert micro SD card in slot underneath Pi card
   - connect Ethernet cable from Pi to Host (or switch)
   - Connect USB power cable from Pi to USB port on host
     - The board should now boot the OS
6. Test connection from Host to Pi
   - $ ping raspberrypi.local
     - should get responses
   - $ ssh -l pi raspberrypi.local
     - enter “raspberry” as password
     - Should see command prompt: pi@raspberrypi:~ $

7. Configure Pi so that a password isn't required on login

   On Localhost:

   - `$ ssh-keygen`
     - `enter a passphase`
   - `$ cat ~/.ssh/id_*.pub | ssh pi@raspberrypi.local 'cat >> .ssh/authorized_keys'`
     - `note: first make sure that Pi already has a .ssh directory`
   - $ ssh -p pi@raspberry.local
     - On first access a dialog asks for passphrase (use same as entered in ssh-keygen)
     - On subsequent logins passphase isn't required

GRIP

1. Reference
   - https://github.com/WPIRoboticsProjects/GRIP/wiki/Running-GRIP-on-a-Raspberry-Pi-2
2. Obtain required library files by pressing the links in the above reference
   - libntcore.so,libstdc++.so.6
3. Download system libraries
   - Create destination directory on pi
     - log onto Pi (ssh -l pi raspberrypi.local)
     - pi@raspberrypi:~ mkdir vision
     - pi@raspberrypi:~ cd vision
     - pi@raspberrypi:~ mkdir grip
   - Install Java 8 on Pi
     - pi@raspberrypi:~ sudo apt-get update && sudo apt-get install oracle-java8-jdk
   - scp libraries from Host to Pi (in a separate Host command shell)
     - $ scp libntcore.so pi@raspberrypi.local:/home/pi/vision/grip/libntcore.so
     - $ scp libstdc++.so.6 pi@raspberrypi.local:/home/pi/vision/grip/libstdc++.so.6
4. Build and Install **mpeg-streamer** to fix reported bug with running USB cameras on Pi
   - Obtain libraries and source code

- Log onto Pi (ssh -l pi raspberrypi.local)
- pi@raspberrypi:~ sudo apt-get update && sudo apt-get install git
- pi@raspberrypi:~ sudo apt-get update && sudo apt-get install cmake
- pi@raspberrypi:~ sudo apt-get update && sudo apt-get install libjpeg8-dev
- pi@raspberrypi:~ cd vision
- pi@raspberrypi:~ git clone https://github.com/jacksonliam/mjpg-streamer.git

- Build from source and install
  - pi@raspberrypi:~ cd mjpg-streamer/mjpg-streamer-experimental
  - pi@raspberrypi:~ make clean all
  - pi@raspberrypi:~ sudo make install
- Test USB Camera connection
  - Connect USB camera to one of 4 USB slots on Pi
  - should see “video0” when camera is connected
- Run mpeg-streamer from a command line to test
  - pi@raspberrypi:~ mjpg_streamer -o "output_http.so -w ./www -p 1180" -i "input_uvc.so -d /dev/video0 -f 15 -r 640x480 -y -n"

    Output:

    MJPG Streamer Version.: 2.0

    i: Using V4L2 device.: /dev/video0

    ..
  - View Stream from web browser
    - Open a tab in Firefox and enter the following URL: http://raspberrypi.local:1180/?action=stream
    - Should see an image from the camera in the tab
  - View Stream on host using a Python script
    - file stream-server.py:

```
#!/usr/bin/python
```

```
import cv2
import urllib
import numpy as np
stream=urllib.urlopen('http://raspberrypi.local:1180/?action=stream')
bytes=''
while True:
    bytes+=stream.read(1024)
    a = bytes.find('\xff\xd8')
    b = bytes.find('\xff\xd9')
    if a!=-1 and b!=-1:
        jpg = bytes[a:b+2]
        bytes= bytes[b+2:]
        i = cv2.imdecode(np.fromstring(jpg,dtype=np.uint8),cv2.CV_LOAD_IMAGE_COLOR)
        cv2.imshow('camera',i)
        if cv2.waitKey(1) == 27:
            exit(0)
```

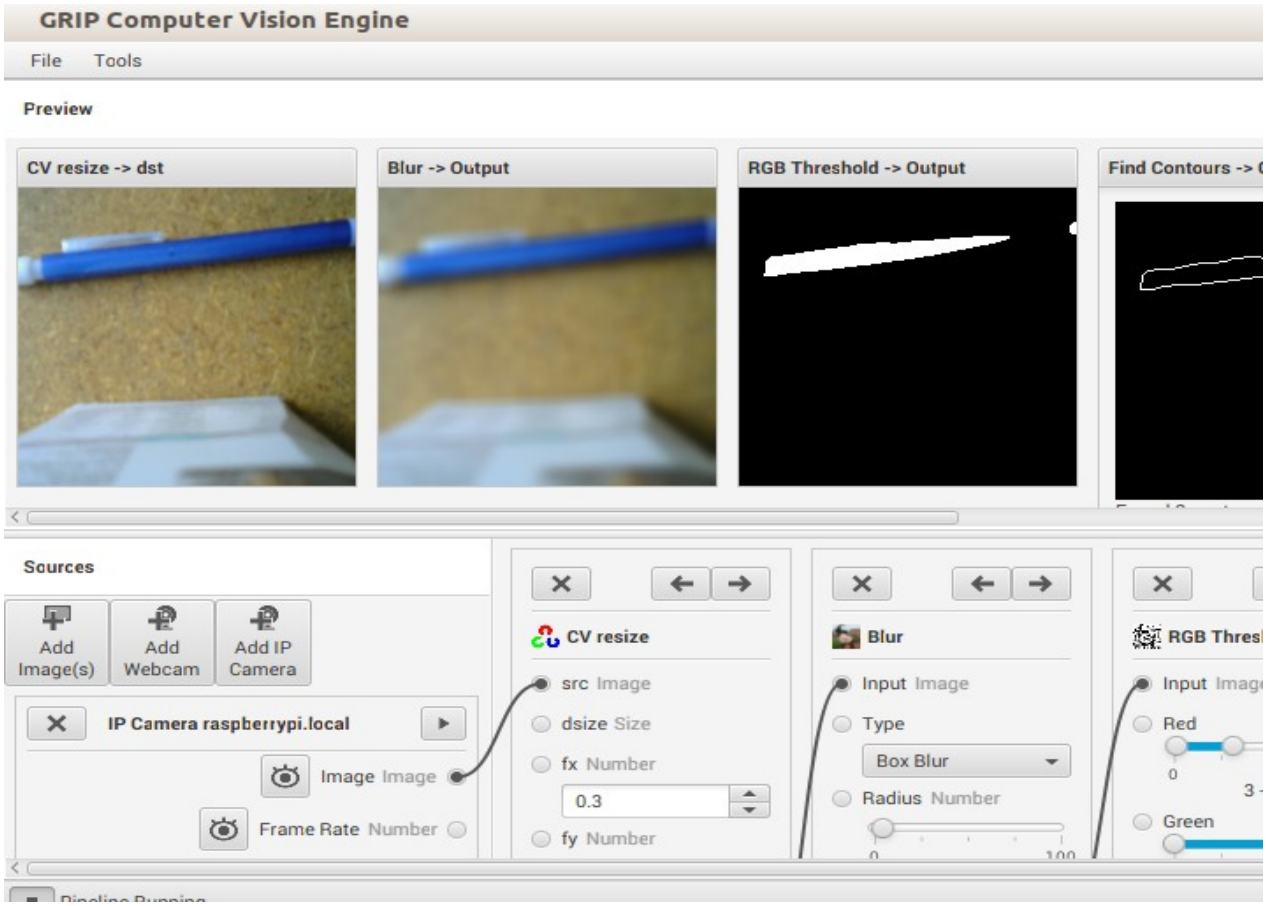
- On Pi, start mpeg-streamer
- On Host, $ python stream-server.py

5. Test Image stream from Pi running GRIP on host

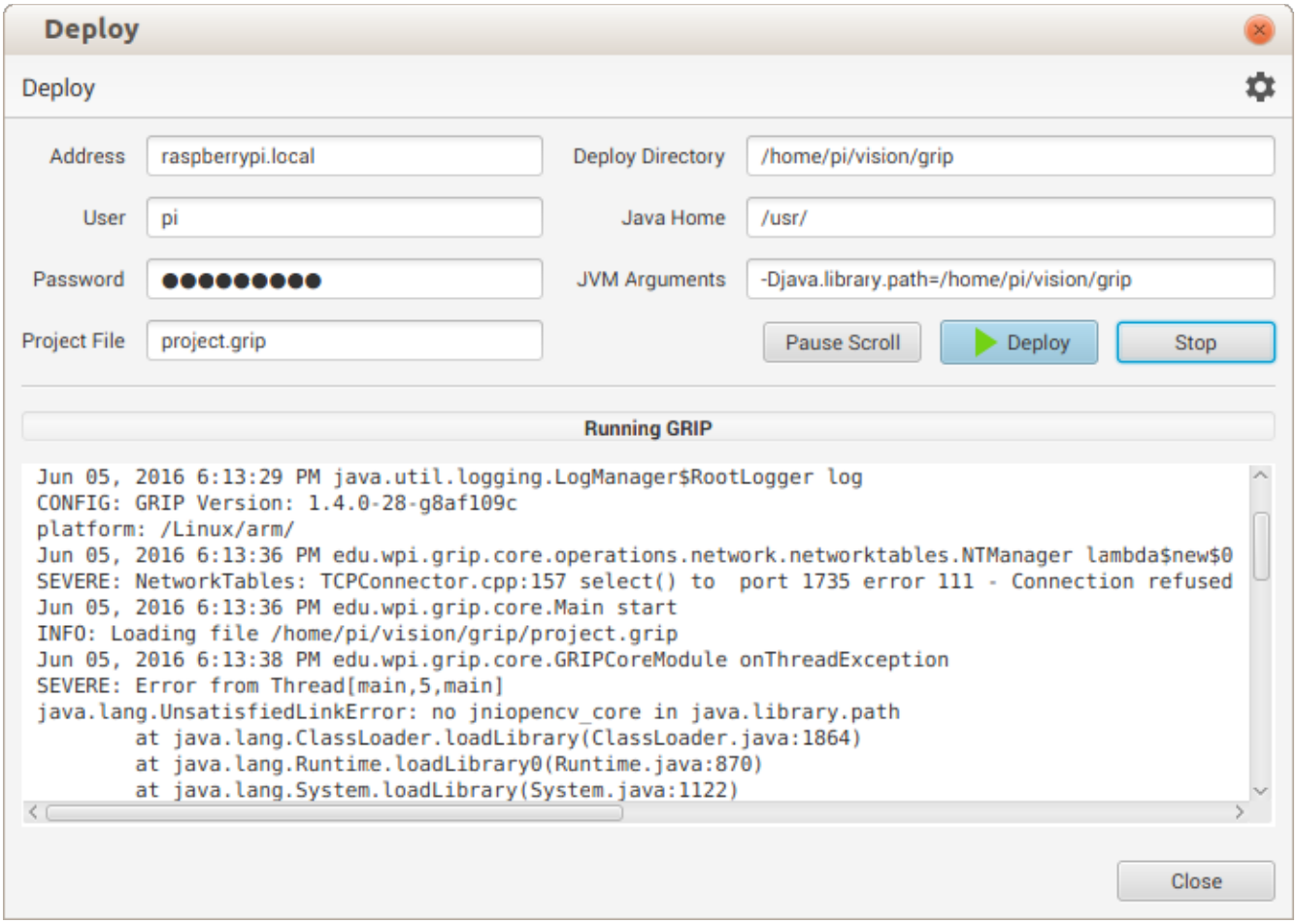
   On Pi start up mpeg streamer as described in the previous section

   - `pi@raspberrypi:~ mjpg_streamer -o "output_http.so -w ./www -p 1180" -i "input_uvc.so -d /dev/video0 -f 15 -r 640x480 -y -n"`

- On Host, start up GRIP with project using one of the methods described above
  - ui/build/install/ui/bin/ui projects/webcam-blue/webcam-blue.grip
- In GRIP UI, replace the WebCam input with “Add IP Camera” input
  - Set URL in Ip Camera dialog to: http://raspberrypi.local:1180/?action=stream
- Should see same processing behavior as when USB camera was connected directly to the Host



6. Verify and try to fix USB camera bug on Pi
    - Was able to get normal USB “WebCAM” GRIP input source to work after installing `fswebcam as described in:` https://www.raspberrypi.org/documentation/usage/webcams/
    - Couldn't see images but got ContoursReport updates in ntables
    - When trying to run mpeg-streamer and a headless GRIP project that was using the webcam simultaneously mpeg-streamer aborted with “illegal image format error”
        - mpeg-streamer worked again once GRIP project was killed
7. Deploy Host GRIP project on Pi (following instructions in ref)
    - On Host start up GRIP and load project
    - Change Source to IP Camera (URL=http://raspberrypi.local:1180/?action=stream) as described in the previous section
    - Set Deploy options as shown



- Pressed Deploy button
  - copies grip.jar (1.4.0-28-g8af109c) and project.grip to ~/vision/grip on Pi
  - fails with error: “no jniopencv_core in java.library.path”
    - without -Djava.library.path get unsatisfied link error for ntcore
  - Same problem observed when running from command line logged into pi
  - libjniopencv_core.so etc. appears to be part of javacv (and javacpp)
  - But haven't been able to locate a pre-built binary for raspberry (arm) so will try to build them from source

8. Build javacpp and javacv (method 1) FAILS

- reference: https://github.com/t-oster/VisiCam/wiki/Raspberry-Pi-installation-on-Raspbian
- `pi@raspberrypi:~` sudo apt-get install maven sed

- git clone https://github.com/bytedeco/javacpp.git
- git clone https://github.com/bytedeco/javacv.git
- build javacpp
  - mvn clean install -f javacpp/pom.xml -Dplatform.name=linux-arm
  - build completes
- build javacv
  - mvn clean install -f javacv/pom.xml -Dplatform.name=linux-arm
  - build fails

9. Build javacpp and javacv (method 2) WORKS
   - reference: http://alltechanalysis.blogspot.com/2015/09/installing-opencv-30-and-javacv-on.html
   - pi@raspberrypi:~ cd
   - pi@raspberrypi:~ sudo apt-get install maven
   - pi@raspberrypi:~ git clone https://github.com/bytedeco/javacpp-presets
   - pi@raspberrypi:~ git clone https://github.com/aravindrajasekharan/JavaCpp-OpenCv-Linux-Arm.git
   - pi@raspberrypi:~ cd JavaCpp-OpenCv-Linux-Arm
   - pi@raspberrypi:~ cp cppbuild.sh ../javacpp-presets/opencv
   - pi@raspberrypi:~ cd ../javacpp-presets
   - pi@raspberrypi:~ export PLATFORM=linux-arm
     - instruction is missing in reference
   - pi@raspberrypi:~ git checkout tags/1.1 -b 1.1
     - instruction is missing in reference
     - compatible with opencv version 3.0
     - note: latest version (1.2) fails to build
   - pi@raspberrypi:~ ./cppbuild.sh
     - builds local opencv 3.0 (succeeds)

- takes ~1/2 hour

- pi@raspberrypi:~ mvn clean install -DskipTests -Dplatform.name=linux-arm
- After ~ 1hour
  - [INFO] JavaCPP Presets ..... SUCCESS [45.900s]

    [INFO] JavaCPP Presets for OpenCV ..... SUCCESS [48:05.816s]

    [INFO] JavaCPP Presets for FFmpeg ..... FAILURE [0.922s]
  - [ERROR] Failed to execute JavaCPP Builder: Could not parse "libavutil/avutil.h": File does not exist
  - But found "libavutil/avutil.h" in several sub-directories
    - e.g. ./opencv/cppbuild/linux-arm/opencv-3.0.0/3rdparty/include/ffmpeg_/libavutil/avutil.h
- However,jni libraries seem to have been built anyway
  - e.g. ./opencv/target/classes/org/bytedeco/javacpp/linux-arm/libjniopencv_core.so exists etc.
- copied /opencv/target/classes/org/bytedeco/javacpp/linux-arm/*.so /usr/lib
- re-ran grip launcher script (start_grip.sh) from ~/vision
  - export LD_LIBRARY_PATH=/home/pi/vision/grip; java -jar /home/pi/vision/grip/grip.jar /home/pi/vision/projects/project.grip &
  - no longer get “no jniopencv_core in java.library.path”
  - but now get the following errors
    - SEVERE: The CV resize operation did not perform correctly.

      java.lang.RuntimeException: /home/pi/javacpp-presets/opencv/cppbuild/linux-arm/opencv-3.0.0/modules/imgproc/src/imgwarp.cpp:3208: error: (-215) ssize.area() > 0 in function resize

      several of these then ..
    - Jun 09, 2016 3:45:09 AM edu.wpi.grip.core.Main onExceptionClearedEvent

      INFO: Exception Cleared Event

      Jun 09, 2016 3:45:09 AM edu.wpi.grip.core.operations.network.networktables.NTManager lambdanew0

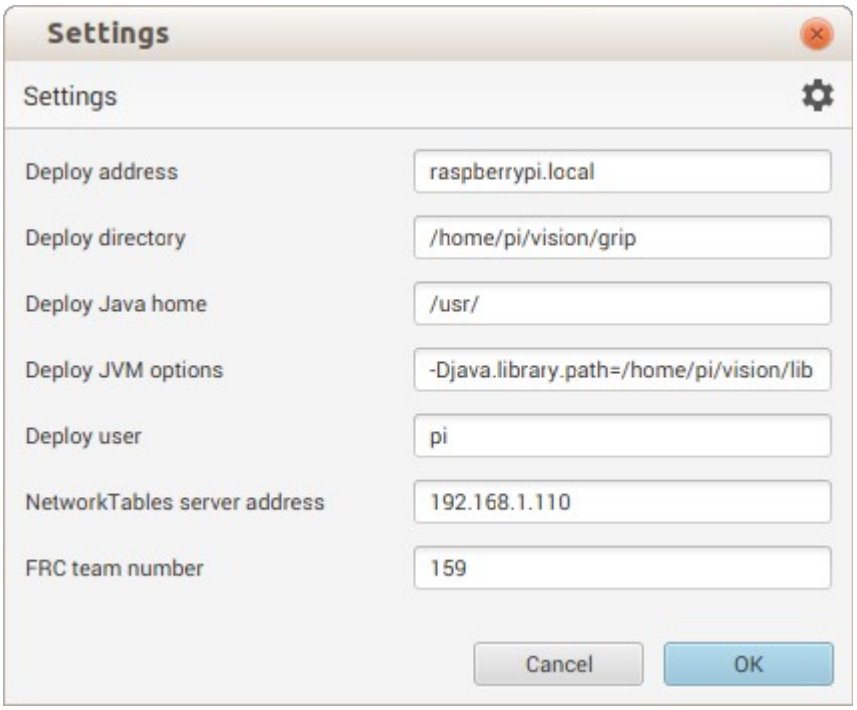
SEVERE: NetworkTables: TCPConnector.cpp:157 select() to 127.0.0.1 port 1735 error 111 - Connection refused

this error keeps repeating until process is killed

- Got “Deploy” in host grip GUI to run (but generate similar errors) by changing JVM Arguments to:

  -Djava.library.path=/home/pi/vision/grip:/usr/lib

  - note: missing jni libraries are now in /usr/lib
- Fix for “connection refused” error
  - On Pi modified project.grip file to change ip of “PublishAddress” element to that of Linux host (as determined by ifconfig)
  - started Network Tables server on host
  - started mpeg-streamer on Pi
  - started start_grip.sh on Pi
  - Now, no longer get connection refused errors but Network Tables GUI shows invalid data in fields “[]”
  - problem was that blue pencil was missing from camera view
  - Repositioned pencil and now see values being updated in Network Tables window !
- Set PublishAddress from GRIP GUI
  - Open “settings” dialog and change “Network Tables Server Address to ip of Host (i.e as determined by running ifconfig)



- can then directly deploy the GRIP project to Pi without needing to hand-edit the project.grip file
- When the “Deploy” button is pressed network tables are updated by the “headless” GRIP project running on the Pi and the image “pipeline” to the UI GRIP running on the host is stopped
- After stopping the headless GRIP project from the “Deploy” dialog window press the small button labled “Pipeline Stopped” in the lower-left corner of the GRIP interface to resume processing from the UI Grip application

10. Summary of how to set up and run a headless GRIP project on the Raspberry Pi
    1. Get Pi resources
        - Install raspian-jessie-lite on Pi SD card
        - Install java 8
        - Obtain or build libjniopencv_core.so etc. for linux-arm
        - place these and downloaded arm versions of libntcore.so,libstdc++.so.6 somewhere accessible from java.library.path (e.g. /usr/lib ~/vision/grip)
        - Build or obtain mpeg_streamer utility and install it
    2. Debug GRIP project on Host
        - Connect USB camera to Pi and point it at a suitable target
        - Run mpeg-streamer on Pi (see above for arguments)

- Start GRIP (with GUI) on Host
- Add IP camera block as GRIP input
    - URL=http://raspberrypi.local:1180/?action=stream
- Add grip filters etc. to isolate regions of interest in camera images
- Add Contours block
- Start up Network Table Server on Host (or run FRCUserProgram which will instantiate a server)
- Add Publish Contours report as last block in GRIP chain

11. From GRIP on Host, Deploy GRIP project to Pi
    1. Connect Raspberry Pi to network
    2. Open Tools-> “settings” dialog
        - change “Network Tables Server Address to ip or DHCP name of Host (e.g. Ubuntu14.local, ip as determined by running ifconfig)
        - set deploy address to raspberrypi.local
        - set user name to pi
    3. Open Tools-> “Deploy” dialog
        - enter password for pi (raspberry)
        - Press “Deploy” button
        - In GRIP UI deploy headless project to raspberry Pi
    4. Verify Network tables publication
        - Open up net-tables viewer as client
        - verify that values for area centerX etc. get updated as before (i.e. when running GRIP from UI in Ubuntu)
            - qualitatively it seems to be the same
            - need to figure out how to measure bandwidth
12. Route network info to Eclipse program running on host
    1. Use Eclipse GRIP Test program described 9.5 above

2. Stop Network Tables server (otherwise get the following error)
    - NT: ERROR: bind() failed: Address already in use (TCPAcceptor.cpp:102)
3. Run project executable from command line (linux_simulate/FRCUserProgram) or eclipse (run as linux_simulate ..)
    - Starts up a new Network Tables server
4. Start up Network Tables as Client to view contours report publications in external window

13. Addendum:
    - To ease the pain of setting up a new SD card that includes support for GRIP I've placed a number of tar files in the Team159:MentoRepository in the “Raspberry” directory
        - see instructions in the README file there for how to install the tar files

OpenCV

1. Version
    - 3.0 (or 3.1)
2. Reference
    - http://www.pyimagesearch.com/2015/10/26/how-to-install-opencv-3-on-raspbian-jessie/
3. Install dependent libraries
    - Log onto Pi (`ssh -l pi raspberrypi.local`)
    - From reference instructions
        - pi@raspberrypi:~ sudo apt-get update
        - pi@raspberrypi:~ sudo apt-get upgrade
        - pi@raspberrypi:~ sudo rpi-update
        - pi@raspberrypi:~ sudo reboot
        - pi@raspberrypi:~ sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev
        - pi@raspberrypi:~ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
        - pi@raspberrypi:~ sudo apt-get install libxvidcore-dev libx264-dev
        - pi@raspberrypi:~ sudo apt-get install libatlas-base-dev gfortran
        - pi@raspberrypi:~ sudo apt-get install python2.7-dev python3-dev

- pi@raspberrypi:~ sudo apt-get install libgtk2.0-dev (do we need this for headless processing ?)

4. Obtain opencv source

   (This procedure differs from reference instructions which uses a tar file)

   - Log onto Pi
   - pi@raspberrypi:~ git clone https://github.com/Itseez/opencv.git
   - pi@raspberrypi:~ cd opencv
   - pi@raspberrypi:~ git checkout tags/3.0.0 -b 3.0.0
     - note: for 3.1 version use: :~ git checkout tags/3.1.0 -b 3.1.0

5. Setup Python (2.7)

   - pi@raspberrypi:~ wget https://bootstrap.pypa.io/get-pip.py
   - pi@raspberrypi:~ sudo python get-pip.py
   - pi@raspberrypi:~ sudo pip install virtualenv virtualenvwrapper
   - pi@raspberrypi:~ sudo rm -rf ~/.cache/pip
   - added to bottom of ~/.profile

     export WORKON_HOME=$HOME/.virtualenvs
     source /usr/local/bin/virtualenvwrapper.sh
   - pi@raspberrypi:~ source ~/.profile
   - pi@raspberrypi:~ mkvirtualenv cv
     - command prompt should now be: (cv)pi@raspberrypi~
     - note: after rebooting etc. to get back into the “cv environment” enter “workon cv” (ctrl-d to exit)
   - (cv)pi@raspberrypi:~ pip install numpy
     - takes about 15-20 minutes to complete

6. Build opencv

   - (cv)pi@raspberrypi:~ mkdir release && cd release
   - (cv)pi@raspberrypi:~ ccmake ..
     - kept most defaults except turned off CUDA and enabled openMP

- (cv)pi@raspberrypi:~ make -j4
  - took ~1hr to build
  - note: after build disk usage increased from 49->93% on 4G SD card -(need bigger card)
    - So purchased a 16G uSD card ($11) and repeated everything up to this point
    - df now shows 25% usage

7. Finishing the installation
   - (cv)pi@raspberrypi:~ cd ~/.virtualenvs/cv/lib/python2.7/site-packages/
   - (cv)pi@raspberrypi:~ ln -s /usr/local/lib/python2.7/site-packages/cv2.so cv2.so
8. Verify installation
   - (cv)pi@raspberrypi:~ python

     >>> import cv2

     >>> cv2.__version__

     '3.0.0'

     >>>
   - Use Ctrl-d to exit Python
9. Python Utility functions
   - Display image data sent from Pi in host window

```
#!/usr/bin/python
# run on Ubuntu host
# opens a window and updates whenever a new image is sent from client

import socket
import cv2
import numpy

def recvall(sock, count):
    buf = b''
    while count:
        newbuf = sock.recv(count)
        if not newbuf: return None
        buf += newbuf
        count -= len(newbuf)
    return buf

TCP_IP = 'Ubuntu14.local' # DHCP hostname or static ip of host
TCP_PORT = 5001
```

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
s.listen(True)
cv2.namedWindow("SERVER")
while True:
  conn, addr = s.accept()
  length = recvall(conn,16)
  stringData = recvall(conn, int(length))
  data = numpy.fromstring(stringData, dtype='uint8')
  decimg=cv2.imdecode(data,1)
  cv2.imshow('SERVER',decimg)
  if cv2.waitKey(25) == 27:
    break
s.close()
cv2.destroyAllWindows()
```

- Send opencv image (captured from USB camera)to Host for display

```
#!/usr/bin/python
# runs on Pi
import socket
import cv2
import numpy

TCP_IP = 'Ubuntu14.local' # DHCP hostname or static ip of host
TCP_PORT = 5001

sock = socket.socket()
sock.connect((TCP_IP, TCP_PORT))

capture = cv2.VideoCapture(0)
ret, frame = capture.read()

encode_param=[int(cv2.IMWRITE_JPEG_QUALITY),90]
result, imgencode = cv2.imencode('.jpg', frame, encode_param)
data = numpy.array(imgencode)
stringData = data.tostring()

sock.send( str(len(stringData)).ljust(16));
sock.send( stringData );
sock.close()
cv2.waitKey(0)
```

Robot Image processing Code

Vision interface class

specifications

1. Front end (API for Robot program) should be the same for whatever back end processing engine is used (e.g. GRIP, ni-vision, native openCV, advanced image recognition tools etc.)
2. Front end API should not be different when compiling for simulation or deploy targets

Front end Interface

1. Configure API with camera and target property information (width, height etc)
    - It may be possible to obtain some camera information automatically
2. Receive target information from back-end in an array of structures or class objects containing:
    - Identity code for each object recognized
    - distance from camera (in feet or meters), left right offset angle, inclination angle (in degrees or radians)

Back end processing

1. Capture net-tables data generated from processing engine (e.g. GRIP)
2. generate target information data structure(s) that contain real world distance and position information etc.
    - Use Configuration information to convert between screen space and real world dimensions

Simulation Tasks

Determine the distance from a target based on the size of the target in an image

1. Distance (inches)=0.866*image_width*target_width/target_image_width;
    - 0.866 = 1/(2*tan(RPD(FOV/2.0))); for FOV = 60 degrees
    - image_width = width of camera image (e.g. 320 pixels)
    - target_image_width = width(pixels)of target in image
    - target_width = assumed physical width of target (inches)
1. for the 2016 tape pattern on a tower, the width of image projection may suffer less distortion (fore-shortening) than the height because of the low angle of view
2. compare distance values calculated from images with those returned using (simulated) lidar sensor
    - Distance values were tested and agreed within ~10%

Determine the shooter angle adjustment based on the vertical position of a target in the image

1. Calculate the vertical error (in degrees) by measuring the displacement of the target center (in pixels) from the midpoint of the image
    - error = 0.5 * image_height - target_centerY * FOV/image_height + v_adjust
        - v_adjust moves the target aiming point up or down from the physical object center
        - For a constant offset
            - v_adjust = 0.886*image_height*target_voffset(inches)/target_distance

- Generally, v_offset should be calculated based on the expected parabolic trajectory of the ball, distance from the target shooter angle etc.

Determine the robot heading adjustment based on the horizontal position of the target in image

1. Calculate the horizontal error (in degrees) by measuring the displacement of the target center (in pixels) from the horizontal midpoint of the image

2. error = target_centerX+h_adjust-0.5*screen_width
   - h_adjust compensates for fixed lateral displacement of the camera on the robot chassis from the center line of the shooter axis
   - h_adjust = 0.886*screen_width*camera_hoffset/target_distance
   - note: horizontal error is sign-reversed from vertical error

Use image data to reliably place a high goal in Autonomous mode

1. In addition to the Gazebo Robot-2016 field simulation A GRIP project is used to extract target data (width height etc.) from the simulated camera and publish it to a network tables server. The following processes are started up (in the indicated order):

   - mpeg streamer
   - Gazebo Robot-field Simulation
   - sim_ds
   - FRCUserProgram (in linux_simulate)
   - GRIP project

2. Data flow Details

   - The Autonomous command group running in FRCUserProgram first moves the Robot in a straight line about ~7 meters though the lowbar and into the opposite courtyard
   - The robot is then turned coarsely towards the target on the left side of the tower (~40 degrees) and the shooter is elevated ~20 degrees (way off target but enough to bring the target within the view of the camera)
   - The image stream generated by the Gazebo camera is published to the LAN by mjpg-streamer (running on the Ubuntu host)
   - The raw images are captured by the GRIP program also running on the Ubuntu host or deployed to the raspberry Pi
   - The GRIP program isolates targets in the image stream and publishes their properties to the LAN (as arrays) using the FRC NetworkTables protocol

- The NetworkTables data is captured by the FRCUserProgram (running in Ubuntu) and is made available to the “Image” subsystem class in the Eclipse project

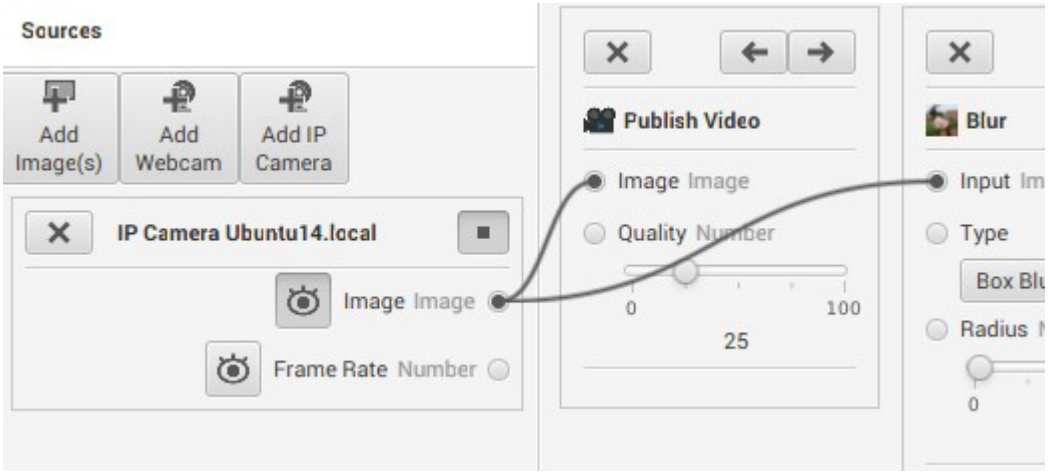
3. In the SimVisionTest Eclipse project a new command “AdjustShot” was created that uses vertical and horizontal offsets in image to center target along shooter axis (by centering target in image)
    - A PID loop is used that maintains the heading to 0 degrees using a simulated gyro
    - Another PID loop is used to stop the robot once the specified distance is reached (using the wheel encoders)
    - The NetworkTables data containing the target center and offset data is used by the AdjustShot command to adjust the position of the shooter and robot to move a selected target to the center of the screen
        - Two separate PID loops are used to adjust the horizontal and vertical position of the shooter
            - the horizontal PID adjusts the drive training
            - The vertical PID adjusts the shooter angle
        - the final adjusted shooter angle is ~36 degrees and robot heading ~50 degrees
    - When the target is centered ball is shot towards the tower goal
4. Results
    - Under “normal” conditions (non-failure modes) a successful goal is scored most of the time (at least 75% ?) using vision assist
        - By contrast, even with the best optimized choices of shooter angle and turn angle a goal is scored only about 20% just using dead reckoning
    - The GRIP project was also deployed to run on the Raspberry PI
        - Similar behavior was seen with small image size (320x240) . In both cases the shot was usually successful
        - for larger images though (e.g. 640x480) the GRIP project running on the Pi failed to keep up and the correction appeared to oscillate
5. Some observed failure modes:
    - Sometimes the shot is aborted because no ball is detected after targeting
        - When the shooter is raised the ball slips out of the holder gate

        - In the original AdjustShot code the holder push wheel speed wasn't changed while targeting to match the new vertical shooter angle (which was set in the previous StepShooterAngle command)
        - AdjustShot was modified so that the holder push wheel speed was also adjusted while auto-targeting was in progress
- Sometimes “no targets” are detected even though targets are clearly showing in the net-tables client window and the GRIP UI
    - Since each target attribute (width, centerX etc.) needs to be extracted as a separate call to the NetTables library function it's possible that the data gets corrupted if the tables are updated between one attribute read and the next
    - The “Image” sub-system code was originally designed to exit (reporting 0 targets) if the array size of all attributes aren't exactly the same during a given table read pass
        - Added a modification that captures all the array values first (including array sizes) and instead of reporting “no targets” if all the sizes aren't the same returns the number of targets found in the last “successful” ObtainTargets function call
- Sometimes during the “AdjustTargets” operation the correction swings too wildly and the camera looses sight of the tower which causes “no targets” to be reported
    - Problem occurs more frequently when GRIP is deployed on the Raspberry Pi
        - In at least one occasion the problem was caused by mjpg-streamer exiting with a failure
        - Problem occurrence lessened by optimizing PID values for Raspberry Pi deployment (using an “#ifdef”)
- Sometimes the shot misses because the robot or shooter report that they are both “on target” (<0.5 degree error) when in fact the error when the shot is taken is actually much larger
    - A Possible reason for this is that the target briefly passes though image center while the PID loops are overshooting (which cause AdjustShot to exit prematurely)
        - Added a modification where two consecutive “on target” results are required to terminate the “AdjustShot” command

Display camera images in SmartDashboard (Java)

1. Display Camera images images using GRIP Publish video Block
    - There appears to be a plugin at https://github.com/WPIRoboticsProjects/GRIP-SmartDashboard/releases that can display unannotated or annotated images in the older

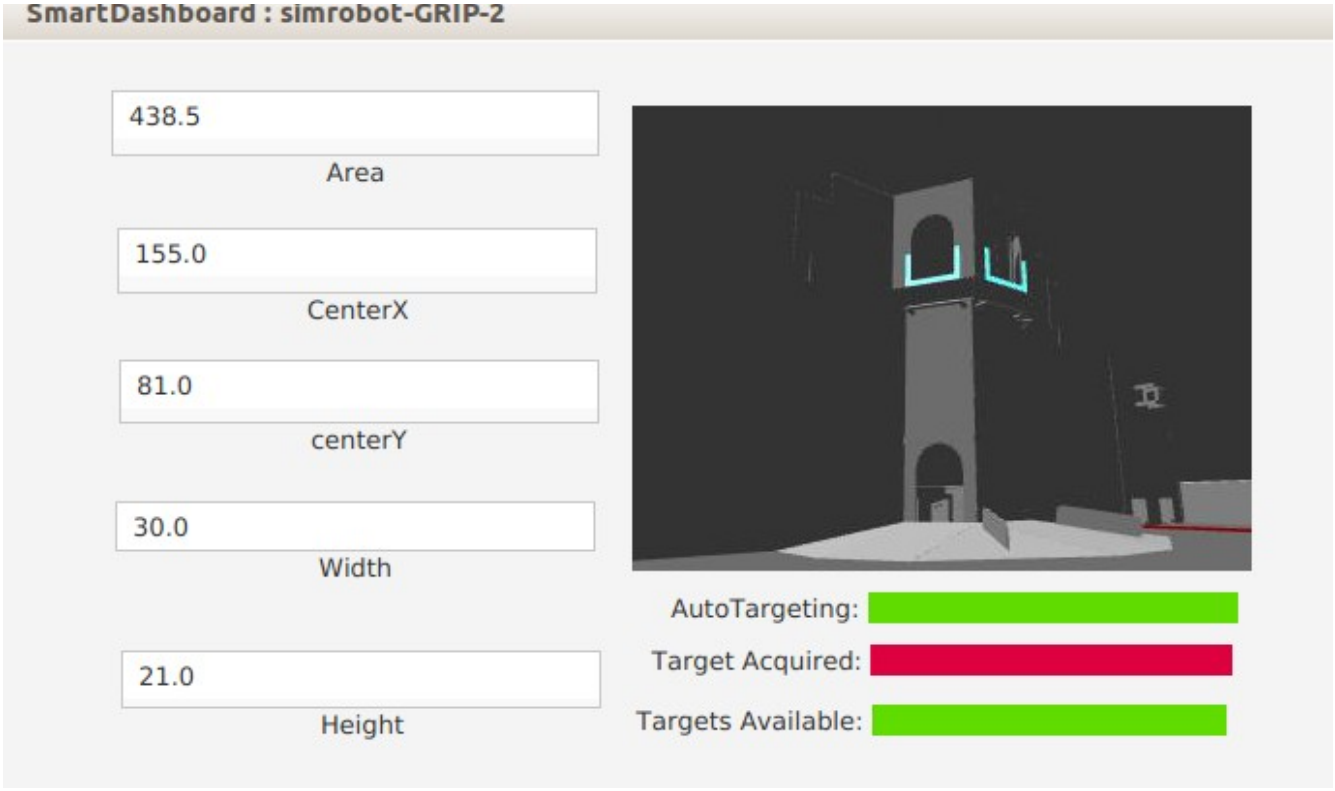
SmartDashboard (not the new JavaFX version)



- Change the port used by mjpg-server and the GRIP ip camera input from 1180 to 5002
    - SmartDashboard appears to be hard-coded to listen on port 1180 which is also (presumably) the port that the “Publish Video” block in GRIP writes to (though I haven't looked at the code yet)
- In SD GUI set View->Editable
- Add a “OutputViewer” to the panel
    - Edit ->Add .. → “GRIP OutputViewer”
    - To change URL right-click in Viewer panel ares and select “properties”
        - Default is URL “localhost” which seems to work for displaying unannoted images

2. Annotate images by drawing bounding boxes around targets
    - Problems
        - The target boxes don't always appear and do not follow the camera after the first frame is shown
        - Can only attach “Publish Video” GRIP block to “IP Camera” source
            - e.g. Output of “Blur block” cannot be used as input of “Publish Video” block
            - GUI wont allow a connection to be drawn
    - Haven't been able to see images from SmartDashboard running on Windows 10

Display camera images in “SFX” (New JavaFX version)



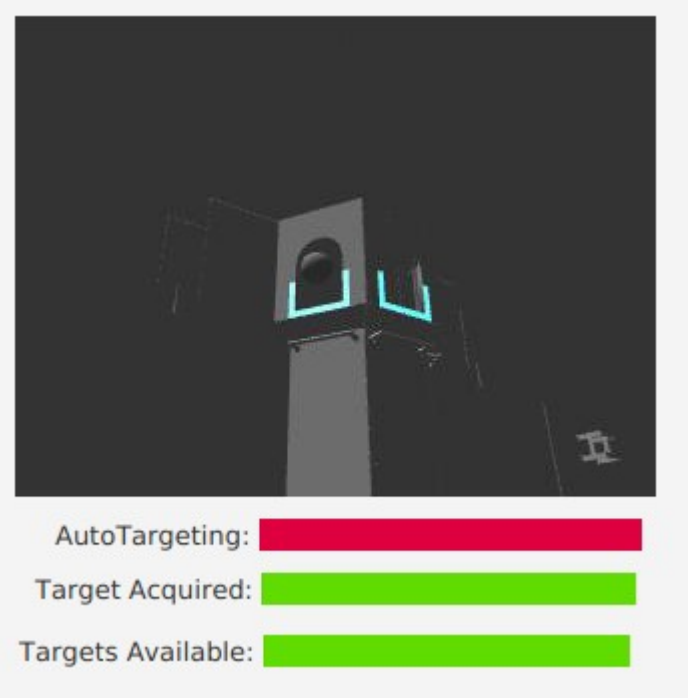
- Add a new Camera object from the “general” list and set it's URL to the video stream being published by mjpg-streamer e.g. : http://Ubuntu14.local:1180/?action=stream
- Problems & Issues:
  - Sometimes there is a very long delay before images start showing up in the Camera Window (or they never appear at all)
  - So far can only see images published from mjpg-streamer (not from the GRIP PublishVideo block)

Support Auto-targeting in Teleop mode

1. Implementation details
   - Added the following functions to “Vision” Subsystem class
     - Set(Get)AutoTargeting
       - Controlled by a joystick button
     - Set(Get)OnTarget

        - Controlled by the auto-targeting system (PID controllers) using target data supplied by GRIP
    - Set(Get)TargetsAvailable
        - Set to true when one or more targets are identified by the GRIP processing system
- Added “Camera” widget to SmartDashboard 3.0 (SFX) to “GRIP/targets” panel as described in previous sections above
- Added SmartDashboard “PutBoolean” code to “Vision” subsystem
    - `e.g. SmartDashboard::PutBoolean("Target Acquired", OnTarget());`
- After Start-up , SFX will auto-populate panels with new Boolean variables
    - Moved these to GRIP/targets panel as shown above

2. Auto-targeting Behavior in Teleop mode
    - In Teleop mode the operator first drives robot to the vicinity of a target using joystick or gamepad controls
    - A gamepad button is then used to raise or lower the shooter until the target is visible in the camera window
    - Once a target (or targets) are showing the “targets available” indicator turns green (this will also enable the “AutoTargeting” button on the Gamepad)
    - When/if the operator presses the AutoTargeting button, feedback from the position of the target in the images will be used to adjust the horizontal and vertical position of the shooter as described previously
    - When the Auto-Targeting PID loops have finished optimizing the “Target Acquired” indicator in the panel will turn from red to green and the auto-targeting indicator will be reset to “off” (red)
    - If the operator then presses the “Shoot” button a ball will be sent towards the target
        - It has been observed that In the simulator a successful shot into the high goal is made almost all the time using auto-targeting



3. Improvements
    - This method still doesn't draw rectangles around the identified targets in the camera window
        - Centered target should be obvious so this is probably not required (but would be a nice feature to have)
    - Since the images in the Camera window are from the input to the GRIP pipeline they don't offer the possibility to show the effect of any GRIP processing
        - On the plus side don't need a “Publish Video” block with this method

Deployment Configurations

Competition

Robot

1. FRC Competition robot or prototype

Processing

1. Image Source
    - Camera: USB, Axis or Pi Camera
    - Connections: Camera to Raspberry Pi via USB slot, Ethernet connector or parallel port
    - Hardware: Raspberry Pi(3)
    - Software: Mjpg-streamer

- Output: Publishes raw image stream to LAN
    - @ http://raspberrypi.local:1180

2. Image processing
    - Hardware: Raspberry Pi
    - Software: Headless GRIP
    - Connections: 2nd enet port on RoboRio switch connected to raspberry Pi
    - Inputs: image stream from mjpg-streamer
    - Outputs: “NetTables data for target parameters (to local network)
3. Robot Control
    - Hardware: RoboRio
    - Software: FRCUserProgram (Deploy build)
    - Inputs: Reads “NetTables” data for targets from LAN
    - Calculates offsets for Shooter and Robot positioning
        - using target, accelerometer and gyro data
    - Outputs:UI data to LAN (NetTables protocol) for input to Driver Station (SmartDashboard)

User Interface

1. Hardware
    - Widows 10 laptop
2. Software
    - SmartDashboard (original application or JavaFX SFX)
3. Connection: Wireless LAN
    - Inputs:
        - Raw Camera images from Raspberry
        - UI data from RoboRio
    - Outputs:
        - Image display in Dashboard camera window

- Targeting Status in Dashboard Widgets (boolean parameters etc)

Simulation

Robot

1. Generated From CAD using Solidworks Exporter
2. Hardware Platform
   - Ubuntu 14 Desktop Laptop
   - Nvidia Graphics card
   - USB gamepad
3. Software
   - Gazebo 6.5 Simulator
4. Inputs to Simulator
   - gzclient publications from Eclipse robot FRCUserProgram
     - issued by running FRCUserProgram
     - value data for simulated motors etc.
     - routed through plugins
   - Joint information from fields in robot sdf file
     - Fixed data generated by Solidworks SDF exporter
   - Mode control from “sim_ds” Java application
     - Proves interface to switch between Teleop Autonomous modes etc.
   - Gamepad controls
     - buttons and joystick knobs
     - Routed to Gazebo from running FRCUserProgram via plugins
5. Outputs
   1. Simulated camera jpg files saved to directory specified in (hand edited) sdf file
   2. gzserver data published to LAN (contains data for simulated encoders etc.)

Processing

1. GRIP running on Ubuntu desktop or Raspberry Pi

User Interface

1. SmartDashboard or SFX

Remote Access (Windows 10)

Setting Up Driver Station to Select SmartDashboard or SFX

SmartDashboard

1. In DriverStation Press Gear Icon
2. Select “Java” as Dashboard Type

SFX

1. Need to edit a startup file
    - Reference: https://wpilib.screenstepslive.com/s/4485/m/26401/l/255410-setting-sfx-to-launch-with-the-ds
    - Change “DashboardCmdLine” in “C:\\Users\Public\Public Documents\FRC\FRC DS data Storage.ini” to point to sfx.jar file (usually in <User-Home>\wpilib\tools directory)
2. Select “Default” dashboard type in driver Station
    - Note: sfx can be really slow to come up

Connecting to the Net Tables Server

1. Whether in Simulation or Deploy Mode the Net-tables Server is hosted on the platform that runs FRCUserProgram (e.g. Ubuntu14 or RoboRio)
2. The Standard Windows Dashboard applications appear to be hard-coded to look for a remote server on the RoboRio with fixed ips (10.xx ..)
3. When either SmartDashboard or SFX is started up from the driver station as described in the previous section they fail to connect to the Net-tables Server when running in simulation mode because the network address isn't what's expected

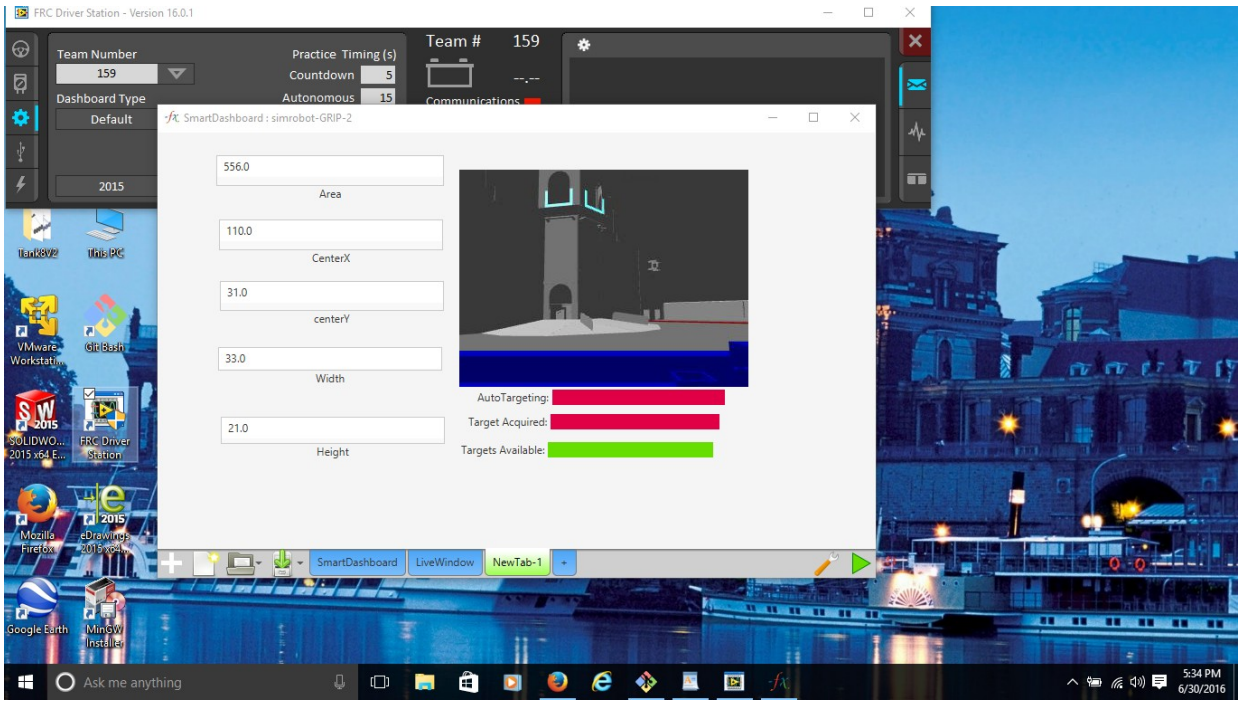
Setting Simulation Mode Connections

1. Connecting to SmartDashboard

    SmartDashboard does provide a command line option that can be used to specify a remote net-tables server

- The syntax is: java -jar <path to wpilib tools>/smartdashboard <ip of remote server>
    - e.g. java -jar C:\Users\<user>\wpilib\tools\smartdashboard.jar 192.168.1.101
        - Regrettably, the ip argument seems to be limited to a numeric string and DHCP names like Ubuntu14.local aren't recognized
    - This method could be used to start SmartDashboard from a DOS command script but haven't determined if that could be configured to run when the user selects the “Java” Dashboard type from the driver station

2. Connecting to SFX
    1. Start sfx.jar with a remote Net Tables server using a command line string
        - Syntax: java -jar <path to wpilib tools>/sfx.jar -nwt://<ip of remote server>
        - e.g. java -jar ~/wpilib/tools/sfx.jar -nwt://192.168.1.101
        - As in SmartDashboard, the “ip” needs to be a network number
            - DHCP names like Ubuntu14.local don't seem to work
    2. Set a new connection Source in the sfx interface
        - Open sfx interface (e.g. java -jar sfx.jar)
        - Press the “+” button
        - Press the “Connections” icon on lower left (looks like a group of arrows)
        - In the “Data Source Selector” panel that comes up press the “Add Data Source” button
        - Set “type” to “NetworkTables”
        - Set “Host” to ip of remote server (e.g. 192.168.1.101)
            - Again, ip must be numeric
        - Press “Save”



Interacting with Gazebo simulation from Windows

Remote Gamepad or Joystick control

Control Device connected to USB port on Windows PC

1. using FRC driver station
    - Would need to fool DS into thinking FRCUserProgram running on remote Ubuntu was actually a Roborio
        - Not sure about how the connection is supposed to work but it probably needs some RoboRio specific service that's not supported in simulation
2. Using sim_ds
    - Would need to run this Java application on Windows and attach it to a remote gzserver
        - Don't know if sim_ds runs on Windows or if it can be attached to a remote gzserver

Through SFX or SmartDashboard Controls

Send messages using Net Tables protocol to change Mode etc. using panel controls instead of a Joystick and sim_ds buttons

1. Create a control to invoke a Command when pressed
    - In c++ code add a command to a Constructor (or RobotInit)
    - e.g. in OI::OI() add: SmartDashboard::PutData("ToggleGate",new ToggleGate());

- When Smart dashboard starts a new Button called “ToggleGate” will appear in the panel
- Disable “edit” mode to use the button
  - In SmartDashbord change the “view” status to “editable”
  - In SFX press the “Play” button (green arrow) in lower right of main window
- Pressing the button caused the command to be executed in the simulation

2. Proposal: Create a Widget to switch between Control modes (Teleop, Autonomous & Disabled)
   - `Write C++ Commands to Set the Operation Mode the state of a variable as described above`
   - `In Robot Constructor add SmartDashboard::PutData commands to attach the commands to buttons`
   - In C++ code create Robot class overrides for `IsAutonomous() & IsDisabled()`
     - `could also include IsTest() override`
     - `note: teleop is the default (there is no IsTeleop function)`
   - `In IsAutonomous() & IsDisabled() functions merge conditions from SFX Commands (set via buttons) with state commands issued from sim_ds (using network protocol)`
     - `e.g. psuedocode for IsAutonomous()`
       - `If SFX Autonomous button is pressed return true`
       - `else return RobotBase::IsAutonomous()`

Advanced Topics

Deep Learning Techniques For Game Object Recognition

Note: It is assumed that the following NVIDIA “Deep Learning” tools have been installed on a high performance CPU/GPU system running Ubuntu 14.04

1. NVIDIA SDK 7.5
2. cuDNN 5.1
3. DIGITS 4.0

A description of the system requirements and installation instructions for these tools is given in a separate document (AdvancedAITools.odt)

Train a simple classifier to recognize objects in a Gazebo simulation

This test will attempt to train a neural network to recognize ball and tote objects randomly placed in the 2015-frc “recycle rush” field. The camera can be mounted on any drivable simulated robot (for convenience the 2016 bot will be used with the camera position raised slightly)

1. Generate a test world file
    - $ export SWMODEL=$SWEXPORTS/2016-Robot_Exported
    - $ export GAZEBO_MODEL_PATH=$SWMODEL:$GAZEBO_MODEL_PATH
    - start up gazebo
        - $ gazebo
    - insert frc_field_2015
    - insert 2016-robot
    - insert and randomly place:
        - 2 gray totes (from 2015 game)
        - 2 balls (from 2016 game)
    - save world as (e.g. vision-test.world)
2. world file tweaks
    - Open “vision-test.world” in a text editor
    - modify sensor block to set image capture rate
        - find: <sensor name='ShooterCamera' type='camera'>
        - add: <update_rate>2</update_rate>
    - modify camera position in sensor block (increase height from ~0.1 to 0.5)
        - <pose frame="">-0.124207 0.5 -0.122065 1.5708 -0 3.14159</pose>
    - modify camera block to save images into a directory (e.g. /tmp/shooter-camera)
        - find: <camera name='Shootercamera'>
        - add: <save enabled='1'> <path>/tmp/shooter-camera</path> </save>
3. Run a gazebo simulation and capture a set of images while driving around the field
    - start up gazebo with the “ vision-test.world” file

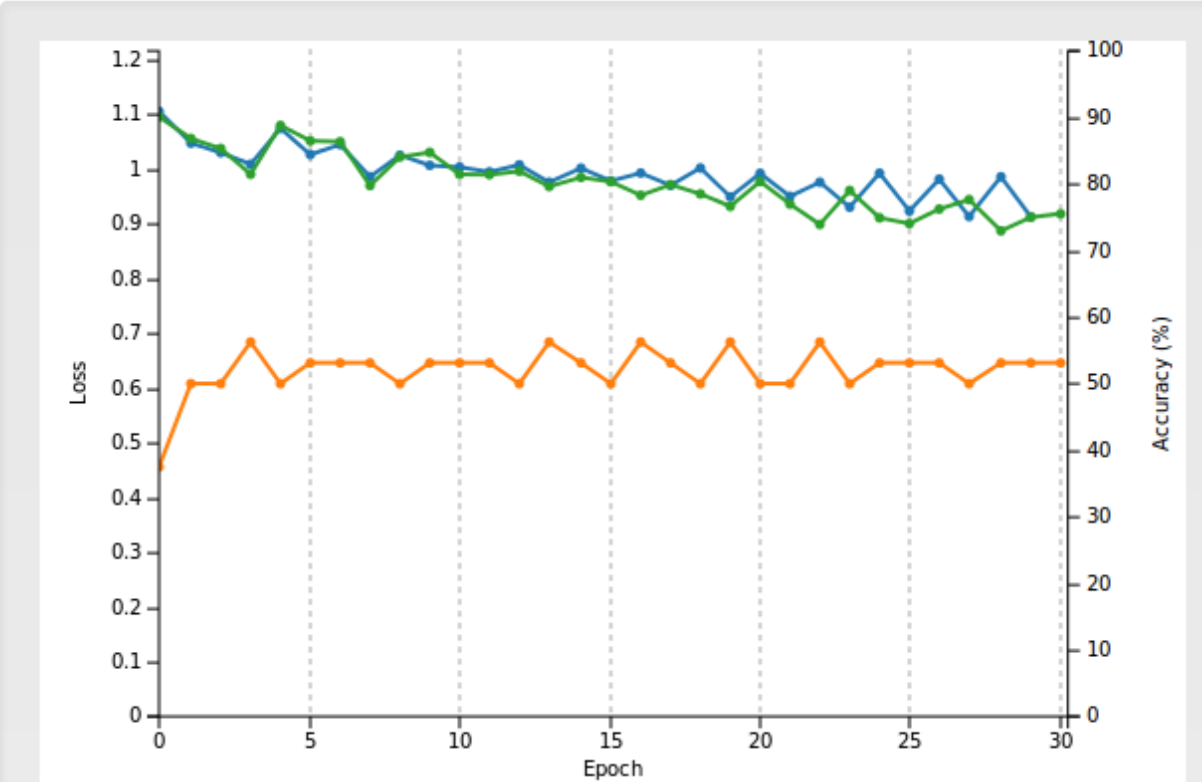
    - $export GAZEBO_MODEL_PATH as shown previously
    - $gazebo vision-test1.world
- start up sim_ds
    - open a new terminal or tab
    - make sure gamepad or joystick is connected to a USB port
    - $sim_ds
- start up FRCUserProgram
    - open a new tab or terminal
    - cd ../MentorRepository/Software/workspace/SimVisionTest
    - ./FRCUserProgram
- Enable “teleop” mode in sim_ds
- “visualize” the gazebo shooter-camera topic
- Using the gamepad, drive the robot around the field
    - try to capture images that primarily only show totes, balls or none of the above
    - a new image will be added to /tmp/shooter-camera at the update_rate specified in the world file
- Exit gazebo,sim_ds and FRCUserProgram

4. Create a set of labeled directories that will be used to generate a DIGITS data set
    - create a parent directory
        - mkdir -p ~/data/vision-test && cd ~/data/vision-test
    - create a sub-directory for each object type
        - mkdir ball tote none
    - populate the sub-directories
        - open a “Places” browser and browse to /tmp/shooter-camera
        - select “View items as a grid of icons”
            - After a short simulation run at 2 frames/s there were ~80 images collected

    - open a second “Places” browser and browse to ~/data/vision-test
    - hand select and move images from /tmp/shooter-camera to ~/data/vision-test/ball that show only a ball in the field (no tote)
    - do the same (to ~/data/vision-test/tote) for images that show only a tote
    - move to ~/data/vision-test/none images that show neither

5. Create a DIGITS data set
    - Open DIGITS in a web browser (http://localhost)
        - If installed correctly the DIGITS web-server should be running after bootup
    - select DIGITS/Datasets
    - select new dataset->classification
    - set image size = 320 x 240
    - set resize transformation = squash
    - in “Training Images” field browse to ~/data/vision-test
    - select 25 % for validation and 10 % for testing
    - uncheck “Separate ..” checkboxes
    - keep other option defaults
    - Set “Dataset Name field” to something (e.g. “ball_tote”)
    - press “Create”
        - After a short delay a new DIGITS dataset should be created
6. Attempt to train a neural network from scratch on this dataset using “Alexnet”
    - In the DIGITS browser window select DIGITS->new Model->classification
    - in the “Select Dataset” list choose the dataset just created (e.g. “ball_tote”)
    - in the “Standard networks”/”Caffe” list choose “Alexnet”
    - Enter a name for the model (e.g. AN on BALL_tote)
    - press “Create”
    - After a short while (2-3 minutes) the training will complete

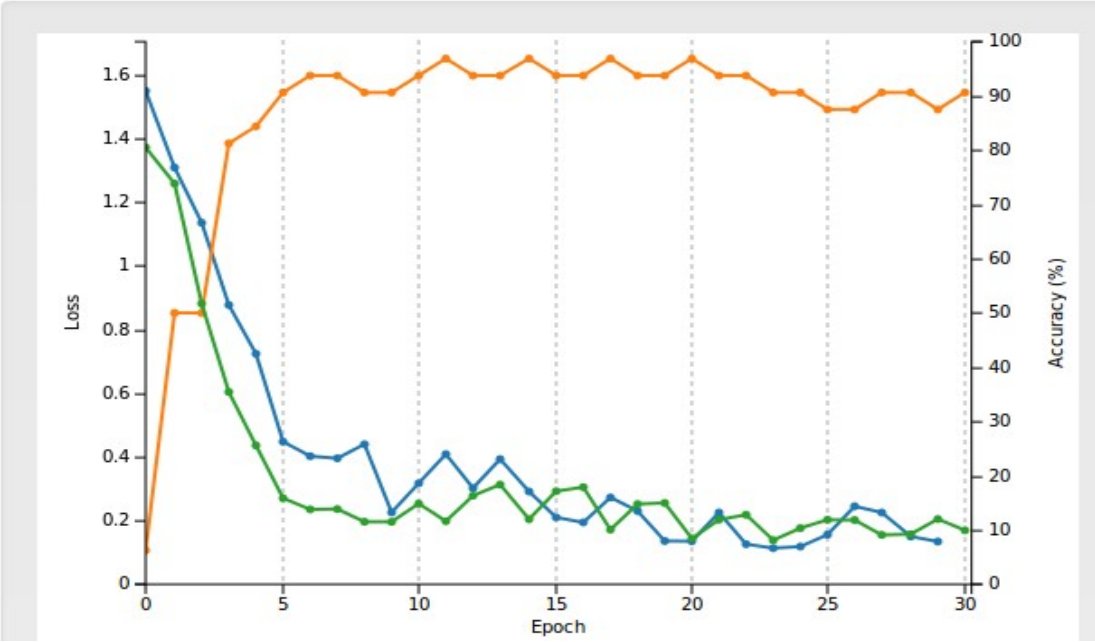
- results:



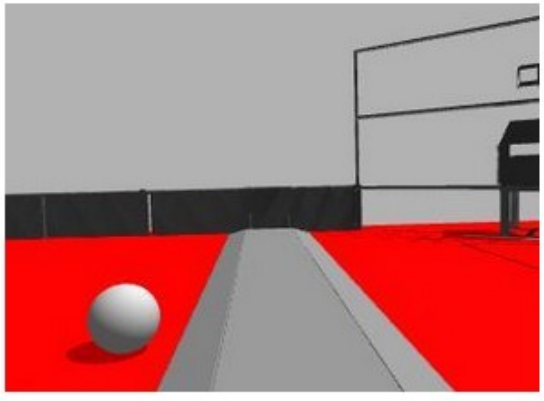
- In this attempt the accuracy is fairly poor (55%) in identifying the 3 catagories (random success would be 33%)

7. Use a pre-trained network to improve results
   - In DIGITS Select Models → Images → Classification
   - Again select “ball_tote” as the dataset
   - keep solver options the same as before except reduce the learning rate from 0.01 to 0.002
   - In the “standard networks” list, again select “Alexnet”
   - press “customize” to the right of the radio-button (this will show the Caffe Alexnet network in text format)
   - in the network text window change all instances if “fc8” to “fc9” (should be 5 of then at the bottom)
   - in the “pre-trained model” entry field browse to the location of a previously downloaded Alexnet “.caffenet” file (e.g. ~/caffe/models/bvlc_alexnet/bvlc_alexnet.caffemodel
   - Give the new model a name (e.g. “pretrained AN on ball_tote”)

- press “Create”
- results

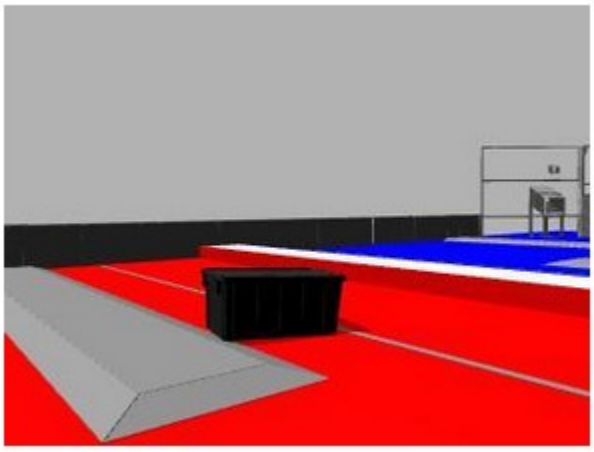


- note, accuracy is now > 90 %
- Test the classifyer on a sample image
- In the model “Test a single image” field browse to one of the images in the 3 sub-directories
- Press the “Classify One” button
- results:



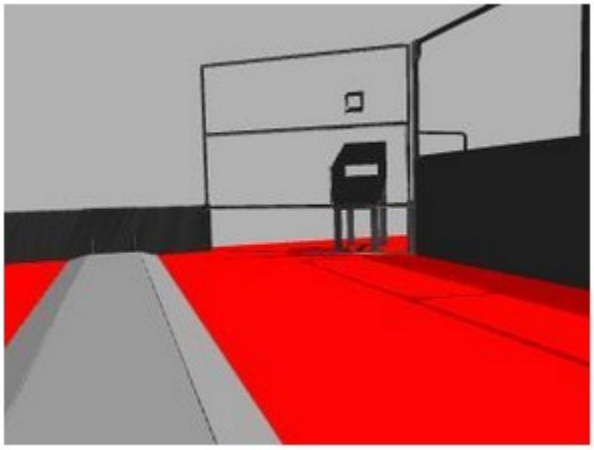
Predictions

ball	99.77%
none	0.16%
tote	0.07%



Predictions

tote	99.21%
none	0.79%
ball	0.0%



Predictions

none	96.85%
ball	2.6%
tote	0.55%

Train a simple classifier to recognize objects in images from a webcam

Use a trained network to find bounding boxes around objects

Real Time Generalized Object Recognition

Use a GPU equipped accelerator board (e.g. NVIDIA TK1) and deep learning tools to identify objects in camera images

Appendix 1: Summary of Build Targets

Mjpg-streamer

Platforms: Ubuntu, Raspberry Pi

GRIP

Platforms: Ubuntu, Windows, Rasperry Pi (headless)

FRCUserProgram

Platforms: Ubuntu, RoboRio

SmartDashboard

Platforms: Ubuntu, Windows

SFX

Solidworks SDF Exporter

Platforms: Windows

sim_ds

Platforms: Ubuntu