It is implied that everything here should work on the master branch unless otherwise stated. Something that we are developing in our own branch but not ready for merging should be pink.

Style guide: Use `${VEHICLE_NAME}` every time it appears.

Other Important docs:

- Making a New Demo Launch file
- Big Finite State Machine
- ROS_Diagram
-

Preparation

This is to be done by the robot's owner.

Hardware:

- Bring the box with all contents given to you, including the pieces that you seldom use (e.g. the charger).
- On the short side of the box there is **your name, and the name of the robot.**
- The robot name is visible on the robot.

```
duckiebot$ sudo pip install graphviz
```

Software:

```
duckiebot$ sudo apt-get install python-sklearn
```

Hardware checklist

- There is a name on the robot
- There are two names on the box (owner/robot)
- Put the robot on the box.
- Put joystick on the box.
- The joystick switch should be on X.
- There is no cap on the camera. If so, remove and put in the box.
- Robot is charging through the charger
    - Check battery level: if less < 3: not ready
-
- (if taking logs) USB Thumb drive plugged in. (not important for April 15)
-

- There is no wifi stick.
- The Pi is powered directly from the battery on one port;
- The Buffalo and the motor use the splitter. The Buffalo is unplugged (both power and ethernet)
- Make sure that the light can turn on on the joystick by clicklng the green button. Or pressing “vibration”.
- The LSD hat (the purple LED board) is present
- LEDs mounted below the top deck (the wires go through the top deck not around) and are out of view of the camera.
- The battery is tightly held with the zip tie.
- Duckie somewhere.
- The camera is not loose
- The wires are neatly tied at the back
- Short usb-barrel connector
- Flexible short ethernet connector
- The total height of everything mounted is less than 6.75 inches.
- The maximum width anywhere is 6 inches.
- The LEDs are individually taped to the connectors
- The LEDs are all white

Software Checklist

- Laptop connected to wireless MIT
- Laptop connected to router via ethernet
- Robot connected to router via ethernet
- You should be able to
    - ping $(VEHICLE_NAME).local
- Make sure that ethernet connection is automatic DHCP
- Open up one terminal for the robotc
- `laptop $ ssh ubuntu@$(VEHICLE_NAME).local`
  It should **NOT** ask you for a password (TODO add link to fix this)
  It should not automatically start byobu.
- `duckiebot $ byobu`
- `duckiebot $ ping google.com`
- `duckiebot $ cd ~/duckietown`
- `Duckiebot $ sudo ntpdate -s time.nist.gov`
    - `Eventually (when master is updated):`
        - `duckiebot $ make fix-time2`

- `duckiebot $ git checkout master`
  - Note: this will become “git checkout openhouse-dp?” eventually.
- `duckiebot $ git pull`
- `duckiebot $ source set_vehicle_name.sh $(VEHICLE_NAME)`
- `duckiebot $ make catkin-clean`
- `duckiebot $ make build`
- `duckiebot $ make unittests-environment`
  - (`make unittests` requires to have the data in DUCKIETOWN_DATA)
- `duckiebot $ make` **`demo-joystick-camera`**
  - `Test robot moves`
  - `Test image on laptop`
- `duckiebot $ make test-led`
  - `You should see the regular pattern (red, green green, blue blue blue, then everyone different.)`

Logging

Mount thumb drive:

```
sudo mkdir -p /mnt/logs
sudo mount -o umask=000 /dev/sda1 /mnt/logs
```

`Try you can write using:`

```
touch /mnt/logs/test_writing
```

Take a log

Take a log using:

```
rosbag record -a --split --duration=120 -o /mnt/logs/<ID>
```

where **<ID>** is what is told you by the director.-

Testing checklist

T-basic: Checking that the software is installed correctly

Run:
`duckiebot$ make unittests-environment`

T-joy: Checking that joystick and actuators work

Requires: Setup step 2.0
Does not require: successful network config / linux laptop

(already part of the “software checklist” above)

On **master** branch

```
duckiebot$ make demo-joystick
```

Push the left joystick forward, the vehicle should move forward.
Push the right joystick right, the vehicle should turn right.

(Check that the joystick is on “X”. Note that sometimes the joystick malfunctions; remove battery and try again.)

T-joy-cam: Checking that joystick and camera works

(This is the one for data collection.)

(already in the software checklist)

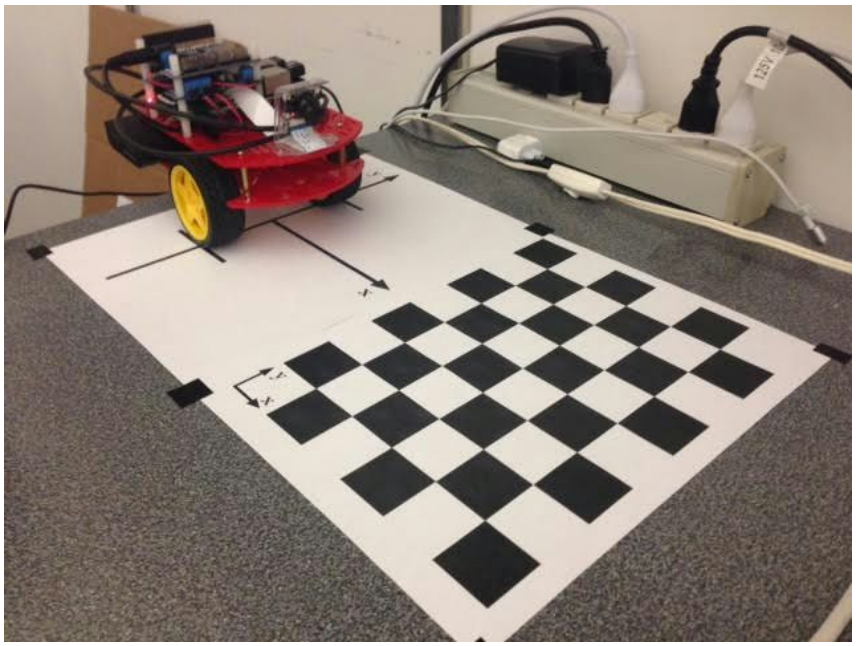
On **master** branch

```
duckiebot$ make demo-joystick-camera
```

T-cam-calib: Checking that camera calibration is okay

Requires: Setup step 2.2.2 complete

Put your duckiebot as shown below:



Note that the axis of the wheels is aligned with the y-axis and camera is looking toward x-axis. Then run the following:

```
laptop $ roslaunch duckietown test_camcalib.launch veh:=${VEHICLE_NAME}
```

If it passes the test, your camera calibration (both intrinsic and extrinsic) is fine. If it fails, please do the camera calibration again as shown in Setup 2.2.2.

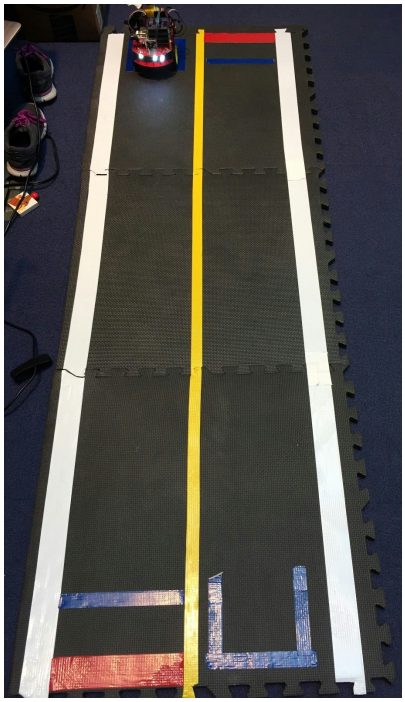
Wheels Calibration Tests

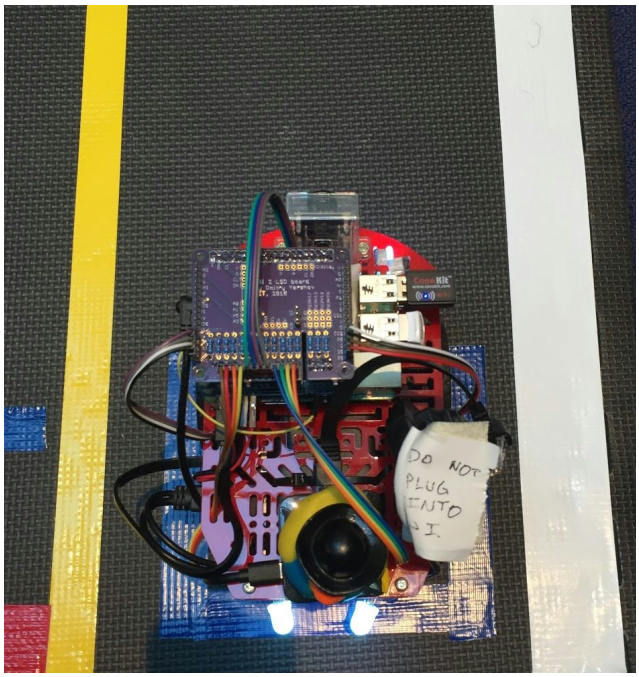
Trim/Gain

How to test:

1) Set your vehicle in the standardized calibration lane, within the black/blue (NOT WHITE/GOLD[1]) taped U. If you run the test, it should reach the black/blue taped line at the end of the lane. See the photos below for the setup:

[1] https://en.wikipedia.org/wiki/The_dress_(viral_phenomenon)





Mat specs (3x1 runway shape):

Red line as close to the edge without crossing the interlocking bits
Blue/Black line 8 cm from red line and parallel to it.
White lines on the edge without intersecting the interlocking bits
Yellow line in the middle of the white lines
Blue/black start position is ~3-4 cm from the edge (not including the interlocking bits)

2) `duckiebot $ byobu`
    a) (Or equivalent) You will need two terminals for this since one command will launch all the necessary nodes + joystick, and the other command will run the test.
3) `duckiebot` $ roslaunch duckietown indefinite_nav_calibration.launch veh:=`${VEHICLE_NAME}`

Note: if there is no kinematics yaml file for your robot, you will see the following warnings, which are expected:

```
[WARN] ... [/<robot>/inverse_kinematics_node]
/home/ubuntu/duckietown/catkin_ws/src/duckietown/config/baseline/calibration/kinematics/<robot
>.yaml does not exist. Using default.yaml.
[WARN] ... [/<robot>/forward_kinematics_node]
/home/ubuntu/duckietown/catkin_ws/src/duckietown/config/baseline/calibration/kinematics/<robot
>.yaml does not exist. Using default.yaml.
```

4) `duckiebot` $ rosrun indefinite_navigation test_straight_line.py veh:=`${VEHICLE_NAME}`
    a) NOTE: This should be run simultaneously with all the nodes launched by the previous command. AKA if you run byobu before

you ran command 2), you can open a new window and run this on your robot as well.

To change trim/etc values:

If your robot veers right in the lane, increase trim value.

If your robot veers left, decrease trim.

If you want to set a trim, where TRIM_VALUE is roughly between -0.1 and 0.1:

```
duckiebot $ rosservice call /${VEHICLE_NAME}/inverse_kinematics_node/set_trim -- TRIM_VALUE
```

For gain:

```
duckiebot $ rosservice call /${VEHICLE_NAME}/inverse_kinematics_node/set_gain GAIN_VALUE
```

Once satisfied with results:

```
duckiebot $ rosservice call /${VEHICLE_NAME}/inverse_kinematics_node/save_calibration
```

Note the "--" which allows negative values of TRIM_VALUE.

The results are saved into a file in `${DUCKIETOWN_ROOT}/catkin_ws/src/duckietown/config/baseline/calibration/kinematics/${VEHICLE_NAME}.yaml`

- You should push the new/updated calibration file to master

TURNS

Use the standardized intersection that has start spots marked in tape, like the mats for the gain/trim test.

1 command on **master**:

1) `duckiebot $ rostest indefinite_navigation calibrate_turn.test veh:=${VEHICLE_NAME} type:={right, left, forward}`

Where "right", "left", "forward" tell which type of turn to test.

2) You can also use the make commands "`make test-turn-left`", "`make test-turn-right`" and "`make test-turn-forward`"

Joystick Buttons Guide

A - coordination signal A
B - coordination signal B
X - no lights
Y - coordination signal C / anti_instagram (ON/OFF) in LANE_FOLLOWING
start - transitions the robot from any state to JOYSTICK_CONTROL
back - transitions the robot from JOYSTICK_CONTROL to LANE_FOLLOWING
Logitech - “E-STOP” cuts power to the wheels (MODE does not change)
On master:

RB - transitions the robot into PARALLEL_AUTONOMY mode
LB - transitions the robot out of PARALLEL_AUTONOMY mode

On so1-devel:

RB - (de)activate verbose mode for line detector
LB - transitions the robot in and out of PARALLEL_AUTONOMY mode

Button stick left - start obstacle avoidance lane following
Button stick right

Demos instructions

Starting procedure for every demo:

- New anti_instagram procedure to learn:
    - Place your robot on the line_detection/anti-instagram calibration tile.
    - Run the command:`$ make demo-lane_following-default`
    - Press RB to activate verbose mode for the line detector.
    - On your laptop run
      `$ source set_ros_master.sh ${VEHICLE_NAME}`
    - On your laptop then open:se
      `$ rqt_image_view`
    - Subscribe to /`${VEHICLE_NAME}/line_detector_node/image_with_lines` (top left drop down menu)
    - Observe the performance of line detector.
    - Press Y to start the anti instragram node.
    - You should see after a couple of seconds the image being color-corrected and the line detections improved.

DP0 Lane Following

Following the starting procedure:
Move the robot onto the lane
Push the start button on the robot

DP1

Run from branch: **master**

TODO: eventually run from dp1-master

Run the following command:

```
$ make openhouse-dp1
```

The duckiebot starts in joystick mode. Set to parallel autonomy mode by hitting the RB button. Remove from parallel autonomy mode by hitting the LB button.

Debugging common problems

(IF YOU FIND A PROBLEM - PLEASE ADD TO LEFT HAND SIDE AND TAG PEOPLE TO FIX):

Observed behavior	Cause of the problem	Solution
Oscillation while stopped	Controller is tuned for a specific velocity (see controller notes from Steven Chen)	Options: 1) 2) 3)
Multiple stops at stop lines		
Stops at stop lines in opposite lanes		

DP2

Run from branch: **master**

~~Run the following command:~~

~~`$ make openhouse-dp2`~~

Launch each detector separately:

```
$ make openhouse-dp2-vehicle
$ make openhouse-dp2-obstacle
```

Starts in JOYSTICK_CONTROL mode. Push the LEFT `JOYSTICK` on the joystick to start the demo.

You should see the robot start to follow the lane. If there are robots in the way it will stop. If there are obstacles detected it will stop (depending on which demo you are running)

Debugging common problems

(IF YOU FIND A PROBLEM - PLEASE ADD TO LEFT HAND SIDE AND TAG PEOPLE TO FIX):

Observed behavior	Cause of the problem	Solution
After pressing start button, robot moves and then quickly stops even though there are no obstacles or cars	It is likely that the obstacle detection module is producing false positives	Step 1) confirm the suspicion by: rostopic echo /${VEHICLE_NAME}/fsm_node/mode If we are in AVOID_OBSTACLE mode then the suspicion is confirmed
The robot continues to drive and cannot be stopped by the e-stop	No commands are being sent to the motors at all. The last one sent is being applied continuously	This is a bug and should never happen. Please post in #dp2-indef_nav on slack
You stop at a stop line	Two options: 1) the red line was detected as a obstacle in which case this is duplicate of row 1. 2) the stop_line_filter caused the stop.	a) The stop_line_filter_node should not be running. b) There will be no stop lines in the duckietown for this demo

DP3

Run from branch: **master**

Run the following command:

```
$ make openhouse-dp3
```

Wait for everything to finish launching.
Place the robot in a lane.
Push the “start” button on the joystick.
You should observe that the robot starts to follow the lane.

Debugging common problems

(IF YOU FIND A PROBLEM - PLEASE ADD TO LEFT HAND SIDE AND TAG PEOPLE TO FIX):

Observed behavior	Cause of problem	Solution
The robot runs very off center in the lane	Your wheel trim is most likely not very good	Run the trim/gain tests
The robot does not stop at the stop line but continues to follow a lane afterwards	The stop line was not detected.	Suggest to hit the e-stop (Logiteck button) and pick up the robot and place in front of the stop line and look at the output of the line detector.
Robot crashes when going through intersection		Run the trim/gain/and turn tests
The robot continues to drive and cannot be stopped by the e-stop	No commands are being sent to the motors at all. The last one sent is being applied continuously	This is a bug and should never happen. Please post in #dp3-indef_nav on slack

SO1 variants

These are special testing instructions to run dp3 with different line detectors.
Run from branch: **master** (?)

variant	*config*	*Command (`make X`)*	*Evaluation (environment: your name: comments)*

Line detector 1 with early April parameters	default.yaml	`make openhouse-dp3-ld1a`	
Line detector 1 (+anti_instragram)	Guy.yaml	`make openhouse-dp3-ld1b`	
Line detector 1 (+anti_instragram)	universal.yaml	`make openhouse-dp3-ld1c`	
Line detector 2 (+ anti_instragram)	ld2.yaml	`To change: make openhouse-dp3-ld2a`	
Line detector 2 + anti_instragram	ld2-universal.yaml	`To change make openhouse-dp3-ld2b`	

DP4

Run from branch: **???**

Run the following command:

`duckiebot $ make openhouse-???` **`laptop, duckiebot $ sudo apt-get install python-matplotlib python-numpy python-pil python-scipy`**

DP5

Run from branch: **???**

Run the following command:

`$ make openhouse-???`

`laptop, duckiebot $ sudo apt-get install python-matplotlib python-numpy python-pil python-scipy`

DP6a

Run from branch: **dp6-integration**

Run the following command:

```
duckiebot $ make build
duckietop $ make build

duckietop $ roslaunch duckietown_demos localization_free_drive.launch
veh:=<vehicle name>
Wait 10 seconds
On a second terminal:
duckietop $ roslaunch duckietown_demos
localization_free_drive_2.launch veh:=<vehicle name>

To adjust the trim and gain:
    rosservice call /${VEHICLE_NAME}/inverse_kinematics_node/set_trim -- 0.0
    rosservice call /${VEHICLE_NAME}/inverse_kinematics_node/set_gain 1.0
    Save with:
        rosservice call /${VEHICLE_NAME}/inverse_kinematics_node/save_calibration
```

If you do this the first time, you will see how it creates a new ${VEHICLE_NAME}.yaml file for your duckiebot in the folder:
duckietown/config/baseline/calibration/kinematics
which you can add and commit to the git repo.

Not Necessary but just in case you need to regenerate the map
roslaunch duckietown_description csv2xacro_node.launch tag_map_csv:=<full path to tag csv file>
tile_map_csv:=<full path to tile csv file> map_name:=<open_house_dp6 OR open_house_dp6_small>

DP6b

Run from branch (laptop and duckiebot): dp6

```
duckiebot $ cd duckietown; make build
laptop $ cd duckietown; make build
```

Run the following commands:

```
duckiebot $ make openhouse-dp6
laptop $ make openhouse-dp6-laptop-<robot>
```

Procedure:

1. Place duckiebot in a lane some distance before but not at an intersection.
2. On laptop, in rqt window:
    a. If you do not see a place to enter source and target nodes, then click Plugins -> Navigation.
    b. If you don't see the graph image, click Plugins -> Visualization -> Image View, and, in the image view pane, select the topic: "/<robot>/graph_search_server_node/map_graph"
    c. Select the start node corresponding to the redline the duckiebot will encounter next.
    d. Select an end node that corresponds to any other intersection redline.
    e. Click plan.
3. Press start on the joystick.
4. Watch the duckiebot follow the path highlighted on the graph.
5. Pressing the joystick override button will cause you to enter joystick control mode and clears the current path, allowing you to plan a new path if desired (useful if the duckiebot crashes).
6. Once a path is completed, a new path can be planned and executed from the current intersection.

Debugging common problems

(IF YOU FIND A PROBLEM - PLEASE ADD TO LEFT HAND SIDE AND TAG PEOPLE TO FIX):

Observed behavior	Cause of problem	Solution
Robot crashes or stops when going through intersection		See dp3, run the trim/gain/and turn tests

---------- things below need to be revised ------

T-joy-net: Checking that network configs + joystick + actuators

Requires: Setup step 2.1 complete

Put joystick receiver in vehicle.

Run:

```
laptop $ roslaunch duckietown joystick.launch veh:=${VEHICLE_NAME}
```

Push joystick buttons. The car should move.

Put joystick receiver in laptop.

```
laptop $ roslaunch duckietown joystick.launch veh:=${VEHICLE_NAME} local:=true
```

Push joystick buttons. The car should move.

T-cam-net: camera + ROS + networks

I want to convince myself that the camera is working and that I can see the output on my computer

Requires: Setup step 2.1 complete

```
laptop $ roslaunch duckietown camera.launch veh:=${VEHICLE_NAME}
```

In a new terminal

```
laptop $ rviz
```

click add in rviz (bottom left)
click “by topic” tab and select the /your_vehicle/camera_node/image/compressed → Image
You should see an Image window in rviz with a live feed from the camera

T-line-detection: I want to convince myself that the line detection is decent

method 1 (visual inspection from log files):

```
Laptop $ roslaunch duckietown line_detection_test1 veh:=ferrari
bagin:= <full_path_to_downloaded_log>
bagout:=<full_path_to_output_bag> verbose:=true
```

let run for X seconds
stop with Ctrl-C

```
$ roscore &
$ rosbag play -l full_path_to_output_bag.bag
$ rviz &
```

in rviz click add
click by topic tab
expand line_detector and click “image_with_lines”
observe on the result that:

1) There are LOTS of detections
2) Predominantly white detections (indicated in black) are on white lines, yellow detections (shown in blue) are on blue lines, and red detections (shown in green) are on red lines

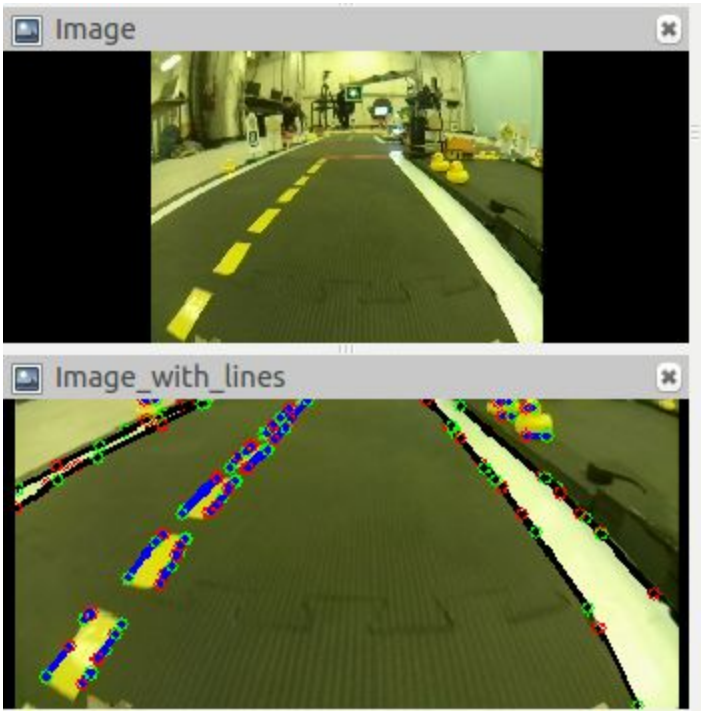
Sample logs:
https://www.dropbox.com/s/mgav6pugm1wd1vz/160122-manual1_ferrari.bag
https://www.dropbox.com/s/i22vz7x4mnxpma9/160122_manual2-ferrari.bag
https://www.dropbox.com/s/vl8d8mo4bpe1a0u/160122_manual3_corner-ferrari.bag
https://www.dropbox.com/s/yz6lfmq9l9tde6g/160122-calibration-good_lighting-tesla.bag

Sample output:
From cmd:

```
[INFO] [WallTime: 1453839555.948481] [LineDetectorNode] number of white lines = 14
[INFO] [WallTime: 1453839555.949102] [LineDetectorNode] number of yellow lines = 33
[INFO] [WallTime: 1453839555.986520] [LineDetectorNode] number of white lines = 18
[INFO] [WallTime: 1453839555.987039] [LineDetectorNode] number of yellow lines = 34
[INFO] [WallTime: 1453839556.013252] [LineDetectorNode] number of white lines = 14
[INFO] [WallTime: 1453839556.013857] [LineDetectorNode] number of yellow lines = 29
[INFO] [WallTime: 1453839556.014539] [LineDetectorNode] number of red lines = 2
[INFO] [WallTime: 1453839556.047944] [LineDetectorNode] number of white lines = 18
[INFO] [WallTime: 1453839556.048672] [LineDetectorNode] number of yellow lines = 28
[INFO] [WallTime: 1453839556.049534] [LineDetectorNode] number of red lines = 2
[INFO] [WallTime: 1453839556.081400] [LineDetectorNode] number of white lines = 13
[INFO] [WallTime: 1453839556.081944] [LineDetectorNode] number of yellow lines = 34
```

[INFO] [WallTime: 1453839556.082479] [LineDetectorNode] number of red lines = 1

From rviz:



method 2: Something more quantitative (to be filled in by Liam or Hang)

cPart 2: I want to convince myself that the calibration of my camera is ok

Requires: Setup step 2.2 (up to Section 3) complete

method 1: compare /image_mono and /image_rec
compare distorted and undistorted lines from a checker board.

As in the extrinsic calibration Setup Step 2.2, put your car as shown in the figure below

Note that the axis of the wheels is aligned with the y-axis.

First, run the following on your external machine (laptop) to run camera node:

```
laptop $ roslaunch duckietown camera.launch veh:=${VEHICLE_NAME}
```

And in another terminal, run `ground_projection` node as

```
laptop $ roslaunch duckietown_unit_test
ground_projection_test1.launch veh:=${VEHICLE_NAME} rectify:=true
```

Make sure that `image_mono` and `image_rect` topics are available:

```
laptop $ rostopic list | grep image_mono

/${VEHICLE_NAME}/camera_node/image_mono
...

laptop $ rostopic list | grep image_rect
/${VEHICLE_NAME}/camera_node/image_rect
...
```

To display image, we can use `image_view` node. Run the following to display the distorted image:

```
$ rqt_image_view
```

Then select the topic `/`**`${VEHICLE_NAME}`**`/camera_node/image_mono`

Old:

```
$ rosrun image_view image_view
image:=/${VEHICLE_NAME}/camera_node/image_mono
```

If you click right mouse button on the image, it saves the image to `frame0000.jpg`
Close the `image_view` node by pressing ‘Ctrl + C’ and rename the image file.

```
$ mv frame0000.jpg frame0000_dist.jpg
```

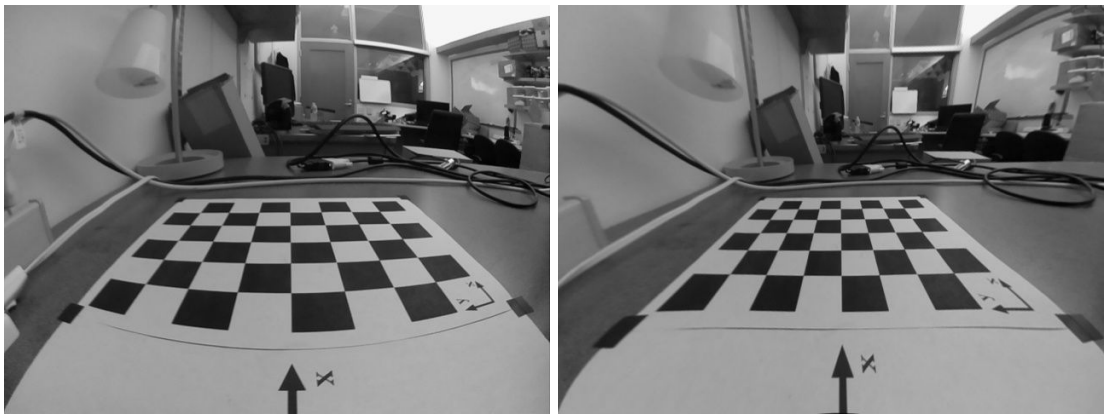
Do the same for image_rect node as

```
$ rosrun image_view image_view
image:=/${VEHICLE_NAME}/camera_node/image_rect
```

After click right mouse button, exit by pressing ‘Ctrl + C’ and rename the image as

```
$ mv frame0000.jpg frame0000_undist.jpg
```

Now compare the checkerboard pattern on two images and note that `frame0000_dist.jpg` image shows severe lens distortion, while `frame0000_undist.jpg` image shows rectified image. If your images look like the images below, your intrinsic camera calibration is done well.



Part 3: I want to convince myself that the reprojection steps work in general

Requires: Setup step 2.2 complete

method 1: fake stimuli

lines in 3D → [projection] → simulated observations of line detections in 2D → [inverse projection] → lines in 3D
(@cchoi says we should use a test image instead of this process)

method 2: using calibration box

```
laptop $ wget
https://www.dropbox.com/s/yz6lfmq919tde6g/160122-calibration-good_lighting-tesla.bag
```

```
laptop $ roslaunch duckietown ground_projection_test2.launch
veh:=tesla bagin:=<full_path_to_160122-calibration-good_lighting.bag>
bagout:=~/duckietown/test_results/ground_projection_test2_out.bag
laptop $ roscore &
laptop $ rosbag play -l
~/duckietown/test_results/ground_projection_test2_out.bag
laptop $ rviz
```

Under “Global Options” click next to “Fixed Frame” (where it says map) at replace with tesla
click add
look at markerArrays and make sure it looks like a square

Paste a sample output here

Method 3: using calibration checkerboard

In this method we use the calibration checkerboard (calibration.pdf) in the extrinsic calibration of Setup Step 2.2 to compare the projected points and the ground truth points

Put your car on the calibration checkerboard as shown in the Part 2 above.

Run the following on your external machine (laptop) to run the camera node:

```
laptop $ roslaunch duckietown camera.launch veh:=${VEHICLE_NAME} raw:=true
```

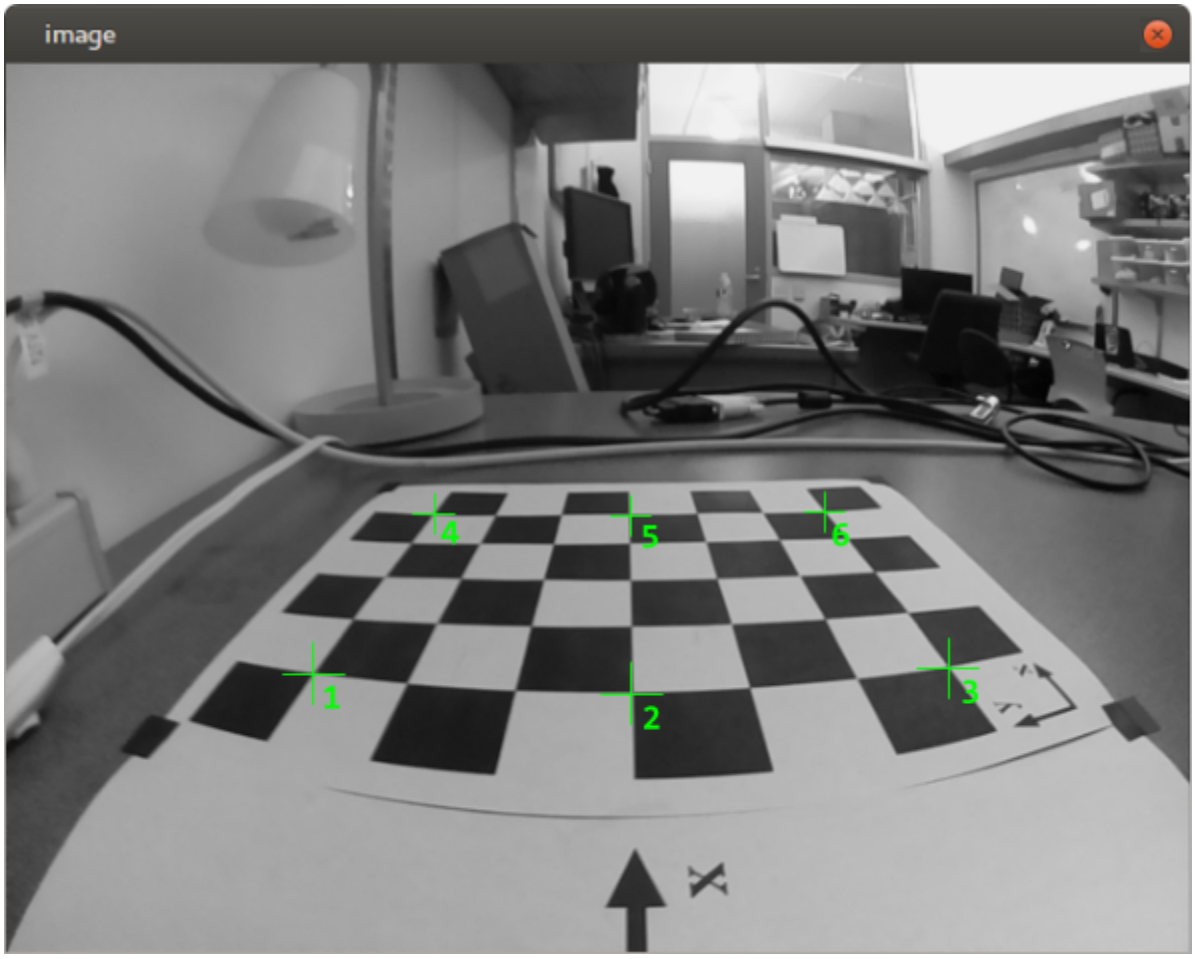
And in another terminal, run `ground_projection` node using:

```
laptop $ roslaunch ground_projection ground_projection.launch veh:=${VEHICLE_NAME}
```

To make sure if the estimated homography is correct, run `test_projection.py` as:

```
laptop $ rosrun ground_projection test_projection.py ${VEHICLE_NAME}
```

It shows an image from the current camera node and returns a ground coordinate if you click a point in the image.



Click the six points as show above. It will returns as follows:

```
image coordinate: (168, 331)
normalized image coordinate: (0.262500, 0.689583)
ground coordinate: (0.192808, 0.094423, 0.000000)
image coordinate: (341, 341)
normalized image coordinate: (0.532813, 0.710417)
ground coordinate: (0.190528, -0.002201, 0.000000)
image coordinate: (511, 328)
normalized image coordinate: (0.798438, 0.683333)
ground coordinate: (0.191026, -0.095562, 0.000000)
image coordinate: (234, 245)
normalized image coordinate: (0.365625, 0.510417)
ground coordinate: (0.304039, 0.085974, 0.000000)
image coordinate: (339, 245)
normalized image coordinate: (0.529687, 0.510417)
ground coordinate: (0.300432, -0.003149, 0.000000)
image coordinate: (445, 244)
```

```
normalized image coordinate: (0.695312, 0.508333)
ground coordinate: (0.297958, -0.090594, 0.000000)
```

The ground truth coordinates of the six points are:

1. (0.191, 0.093, 0.0)
2. (0.191, 0.0,   0.0)
3. (0.191, -0.093, 0.0)
4. (0.315, 0.093, 0.0)
5. (0.315, 0.0,   0.0)
6. (0.315, -0.093, 0.0)

Compare the estimated ground points with the ground truth. Note that the closer points (1, 2, 3) have smaller errors (about 1~2 millimeter), while the further points (4, 5, 6) have relatively larger errors (about 1~2 centimeter).

If your estimated ground coordinates are similar to the ground truth coordinates, your intrinsic & extrinsic calibration is done properly. Press 'ESC' to exit the testing.

Part 4: I want to convince myself that the lane filter works well

LP

method 1:

All the rest require Part 3 to be complete (you have a working homogrpahy)

method 2: using special stimuli with robot placed at specific poses (with robot X using log file Y and line filter revision Z and callbration revision)

> robot = known good calibration
>
> robot_pose_0_0.bag
> robot_pose_10_0.bag
> robot_pose_20_0.bag
> robot_pose_20_0.bag

method 3: using fake stimuli

phi, d → [world model] -> lines in 3D → [projection] → simulated observations of line detections in 2D → [lane filter pipeline] → phi, d

method 4: using ground truth from motion capture

we should be careful about the map - we only need 4 markers for the map - say the 4 corners

Part 5: I want to convince myself that the lane controller works well, also on my robot (follows a lane given that Part 4 is achieved)

Procedure: build a 6 tile highway. You put the robot at the beginning. You launch the file “????.launch”. You expect to see the robot coming out the other way.

Part 6: I want to convince myself that the lane controller + stop at the stop line works well

Procedure: build a 6 tile highway with final red line. You put the robot at the beginning. You launch the file “????.launch”. You expect to see the robot coming out the other way and then stop at the end.

Part ?: I want to convince myself that the NAPL works

TODO: MN

Part ?: I want to convince myself that the coordination works

TODO: DH/HZ

How to verify calibration using calibration stimulus

This document describes how to run the line detection pipeline, reprojection, etc, until the visualization in ROS. This is the procedure used to make sure the calibration is correct.

The expected result is: when I put the car in front of the calibration box, I can see the reprojection of the lines in rviz in world frame appearing as orthogonal patterns.

…

1. Run roscore locally.
   `roscore`
2. Run a rosbag with camera video /$VEHICLE/rosberrypi_cam/image_raw,
   `rosbag play -l <xxx.bag>`
   or, run camera node remotely on a vehicle (note that in this case you should be in the 'duckietwon' network),
   `roslaunch duckietown camera_remote.launch veh:=$VEHICLE`
3. Now it is time to run line_detector node.
   `roslaunch line_detector line_detector_node.launch`
4. Run ground_projection node so that segments are projected onto world coordinates.
   `roslaunch ground_projection ground_projection.launch`
5. Run duckiebot_visualizer node, to convert segment list to markers.
   `roslaunch duckiebot_visualizer duckiebot_visualizer.launch`
6. Visualize all the output in rviz.
   `rviz`
   add by topic:
   raw image '/$VEHICLE/line_detector/image_with_lines'
   image marked with detected lines '/$VEHICLE/line_detector/image_with_lines'
   segments marker '/$VEHICLE/segment_list_markers/MarkerArray'
7. FURTHER: run lane_filter node
   `roslaunch lane_filter lane_filter_node.launch`
8. FURTHER: run lane_controller node
   `roslaunch lane_controller line_controller_node.launch`

How to verify that the all of the perception things work

This is to test everything works together and then visually inspect the result.

1. Choose a log on which to test everything. Call it <bag_in>
   a. This bag need to contain only the input data (compressed camera image (and info), output motor commands, possibly IMU).
2. Run roscore on your computer and make sure the ROS_MASTER_URI points to your computer.
3. You can test the whole pipeline by running this one command:

```
roslaunch duckietown test_perception_pipeline.launch bag_in:=<bag_in> bag_out:=<bag_out>
```

4. Open <bag_out> in Rqt_bag. Do visual inspection.
5. Analytical methods for generating graphs/metrics of success.

LP: TODO: finish this visual inspection checklist. Visual inspection checklist:

- Verify that line detections are decent (qualitatively).
    - add a screenshot of what you expect this to look like
    - save a processed bag where the output is good
    - quantitative?
- Verify that line detection messages have max latency = 120ms compared to image messages (visual inspection is fine)
- Verify that signs are detected.

```
test_perception_pipeline_all.launch -> all of it
test_perception_pipeline_L1.launch -> just line detection
test_perception_pipeline_L2.launch -> just line detection + lane filter
test_perception_pipeline_L3.launch -> just object detection
test_perception_pipeline_L4.launch -> just LED detection
test_perception_pipeline_L5.launch -> just LED detection + vehicle_filter
```

TODO: verify that all the processed data is also published as an image, like the line detection

TODO: additional scripts

CLONE WORKING IMAGE

Clone Working SD Card Image to work with your robot:

1. Grab Shamrock's SD card and plug into the master slot on the SD Card Cloner
2. Place one or more SD cards into the slave slots (NOT LABELLED MASTER)
3. Plug in the power cord in the SD card cloner
4. Power on the SD Card Cloner by pressing the bottom right black button
5. Highlight the 'Clone' option with the Up and Down arrows to the right of the screen
6. Select 'Clone' by pressing the 'Ent' button
7. When cloning starts you will see many blinking lights and a % copied on the screen
8. A good clone will result in the screen saying NG: 0

Modify Cloned SD Card to work with your robot

1. Plug the cloned SD card into your robot's raspberry pi unit
2. Plug in a keyboard and monitor to your robot
3. Power up the raspberry pi
4. Log on to the robot using the monitor and keyboard

Duckiebot log on user: ubuntu pword: ubuntu

5. Edit /etc/hostname and put “duckiebot” instead of “shamrock”

duckiebot $ sudo nano /etc/hostname

6. Edit /etc/hosts and replace “shamrock” with “duckiebot”

duckiebot $ sudo nano /etc/hosts

7. Remove the persistent file

$ sudo rm /etc/udev/rules.d/70-persistent-net.rules

8. Reboot your bot:

$ sudo reboot

If successful, you should see your new hostname:

Ubuntu 14.-4.3 LTS duckiebot tty1:
duckiebot login:

At this point you should also be able to ping and ssh into you duckiebot from your laptop if networking was setup prior, otherwise proceed to the setup of the Buffalo.