Chapter 3

Becoming the navigator of Duckietown

For this lab we are going to build on top of what we were working on previously by done on lab 2 where this time we are mapping the whole duckietown to better achieve localization of the duckiebot in world frame. But first and foremost we should start to detecting some landmarks within duckietown. These landmarks will be april tags that are indicating different street signs and the university's logo. So the first task we have to access our robots camera to be able to see these april tags. Then we have to do some image manipulation to undistort the image created by our fish-eye lens, afterwards the undistorted image will convert into a monochrome image to facilitate our april tag detection. Finally by using the duckietown's april tag detector we are able to achieve information on the april tag that the robot is visualizing. This can be seen not only by image 1 but demonstrating the functioning program through video 1 presented below.

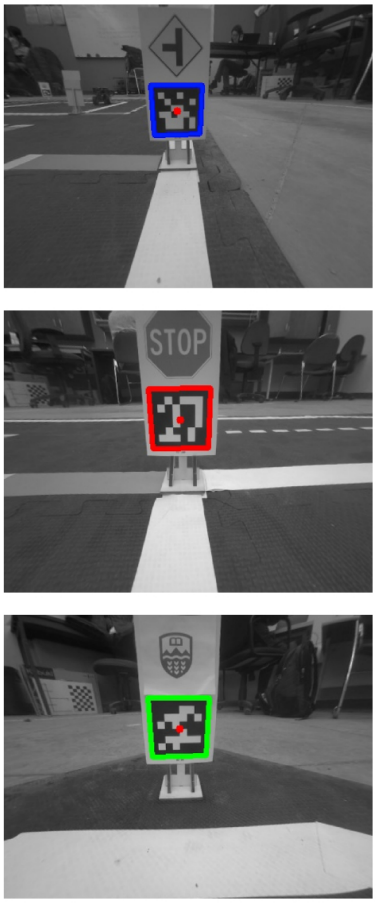


Image 1 - Color indication of the three different tags present in duckietown.

Questions related to the first task:

- What does the april tag library return to you for determining its position?

  The library offers us the various matrices, but from the information we obtain, we utilize the matrices named pose_t to determine its position.

- Which directions do the X, Y, Z values of your detection increase / decrease?

  The x value in pose_t increases when the april tag is moving to the right and decreases when moving to the left. The y value in pose_t increases when the april tag is moving down and decrease when the april tag is moving up. The z value in pose_t increases when we move away from the april tag and decreases when we approach the april tag

- What frame orientation does the april tag use?

  The frame orientation being used by the april tag is represented in the following image (drawn image)

- Why are detections from far away prone to error?

  Our detections from far away are prone to error because at large values of z the image we obtain from our camera will be a lot noisier due to the fact a lot less pixels are being use to represent that april tag. A solution to increase the depth at which can detect an april tag would be to use a camera has a greater pixel count and larger frame of view (ex. 1920*1080)

- Why may you want to limit the rate of detections?

  This is due to the amount of information that is being determined by our april tag detector that can slow our processes or nodes running, which could, for example, slow down the detection for the lane-following task. Since we don't need constant information on our april tags, therefore we can limit are detections. In our case we try to detect an april tag two times every second.

Extras:

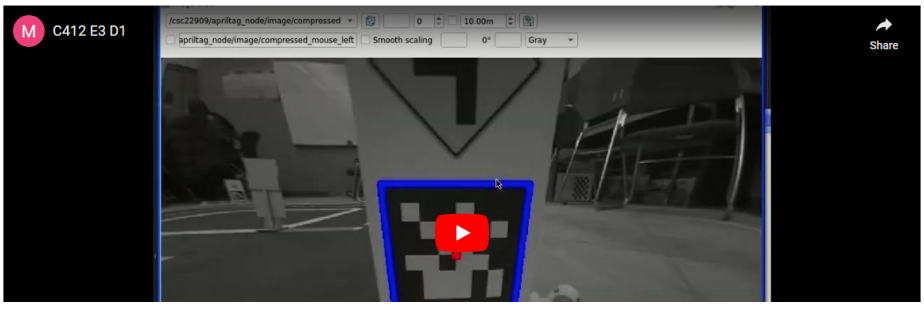
* Question: What does the april tag library return to you for determining its position?

```
[Detection object:
tag_family = b'tag36h11'
tag_id = 58
hamming = 0
decision_margin = 49.139190673828125
homography = [[ 1.74756825e+01  8.64495370e+00  2.45873490e+02]
 [-3.28097126e+00  2.72742042e+01  1.61800476e+02]
 [-1.74551624e-02  2.76566338e-02  1.00000000e+00]]
center = [245.87348955 161.80047645]
corners = [[226.81091309 184.05270386]
 [269.24740601 183.91748047]
 [266.73721313 137.44572449]
 [222.01776123 139.22756958]]
pose_R = [[ 0.99545525  0.02066874  0.09296049]
 [ 0.00841232  0.95326151 -0.30202933]
 [-0.09485823  0.3014387   0.9487553 ]]
pose_t = [[-0.09160954] [-0.12203697] [ 0.42918055]]
pose_err = 6.715101430686409e-07]
```

* Question: Try which tag size the `dt_apriltags` expects?

Tried:

- 0.0645: from the edge length, big mismatch
- 0.0445: from the "distance between detection corners" described in [here] (https://github.com/AprilRobotics/apriltag/wiki/AprilTag-User-Guide#pose-estimation).
- 0.091: from the diagonal distance, works after adding the 106mm distance between robot and calibration chessboard coordinate.



Video 1 - Demonstration of the april tag detection through the robots camera feed

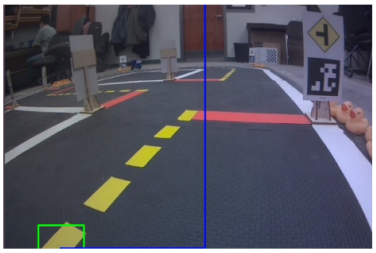


Image 2 - Image manipulation of the object detected with the largest contour

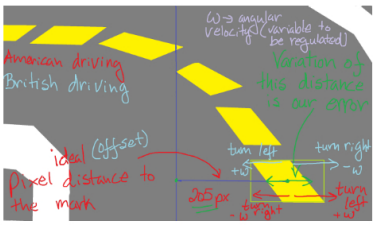
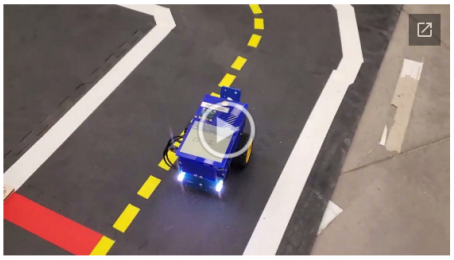
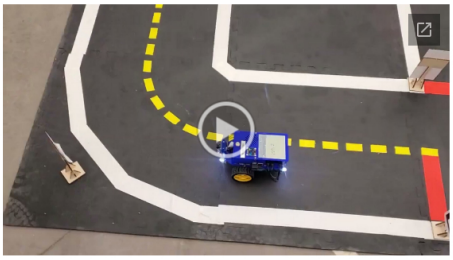


Diagram 1 - Explains how the algorithm used works

Once we are able to detect all the landmarks in our duckietown, the next step is for our duckiebot to drive within the lanes. For this part, we had to incorporate some functions of the OpenCV library to do some image manipulation to measure our current pixel distance from the middle of our camera frame. Such a camera frame should keep us within the lines of our lane the best it can. The part that took the longest to deal with was adjusting the proportional gain for our robot. This constant multiplied by our error (current pixel distance our lane dash from the middle of the frame - our target pixel distance that we want to maintain [offset]) gives us the value we need to set our omega (angular velocity) such that we can turn during the corners and adjust ourselves within the lane. The P variable mentioned previously comes from the concept of P.I.D controllers, which is a mechanism that assists us in regulating a process variable such as the angular velocity in our case. The P.I.D (proportional integral derivative) controller helps stabilize and maintain the most accurate value for our process variable (in our case, omega) through a close feedback loop. In our exercise, it was only necessary to use of the proportional variable (P. gain). For our use case, there is an argument to use the derivative, but it would not create an overall. One of the issues that kept occurring was network lag which in certain situations made the task at hand difficult because it would disrupt the detection process, supplying incorrect readings to the P controller. The only solution possible was to find a time when there weren't many robots connected to the network.



Video 2 - American Driver Lane Following



Video 3 - British Driver Lane Following

Questions related to the second task:

- What is the error for your PID controller?

  The error of my PID controller is simply subtracting the distance of the object detected (yellow dash line) with the target distance in pixels (offset) position. This target distance is the ideal pixel distance that we wanted from our camera from the middle of the lane (the video frame middle) and the closest dash.

- If your proportional controller did not work well alone, what could have caused this?

  Our proportional controller was sufficient to complete the exercise but there is a deviation of our detection once we reach an intersection creating an argument to potential use the D variable. But mostly could be flaw in the program design since the TA Justin demonstrated his work and he utilized only the P term.

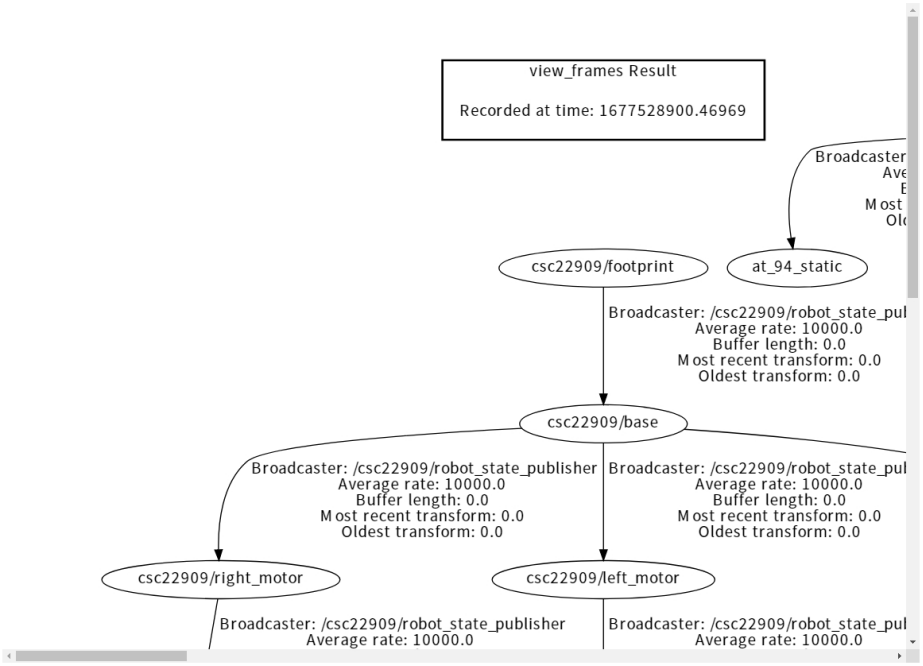
- Does the D term help your controller logic? Why or why not?

  It would make the control of our angular speed a lot smoother, especially some of the jerk motions that would occur in the T intersections, but those seemed to be more flaws of design the need for a derivative term in our controller.

The final task for this lab was localization using sensor fusion, meaning that from a mix of sensors and mapping, we can localize objects of interest (robot and april tags). This part is quite involved in terms of information, so it did give my partner Marcus some trouble, but he managed to pull through with great work. To initiate this part, we first had to set our robot frame into the RViz. We set it up by using the dead reckoning package given and modifying it to establish our parent frame (world frame) and child frame (robot odometry frame). This is the terminology used by Rviz, as we will see later. Once that is set up, Marcus sets up the necessary components to visualize our robot moving in circles in RViz. The next step involves placing a static frame of our april tags into our world frame, using TF visualization in RViz, then using the dead reckoning node to update the coordinates [X, Y, Z, Theta] of our robot through a service call. The use of the tf2 library helps us keep track of all of the coordinate frames, like static frames, such as our landmark or dynamic frames, like the joints of a robotic arm. This permits the user to use these points and vectors to transform them. The library also assists us in maintaining a relationship between these coordinates and the transform tree graph we will see down below.

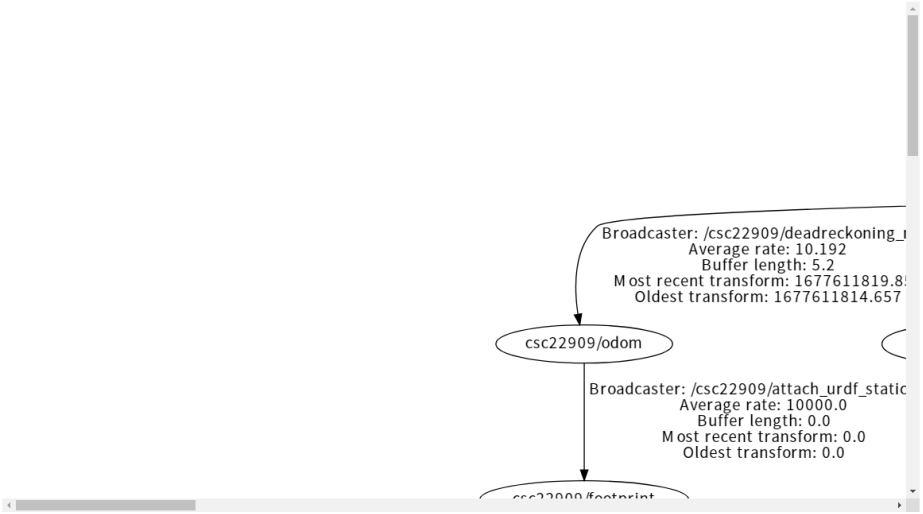
Video 4 - RViz displaying the positioning of our robot and april tags within our world frame.

The next step was to generate a transform tree graph by utilizing the URDF (Unified Robotic Description File) such that from the transforms that are broadcasted by our duckiebot we can figure out the root/parent transform of the relationship between robot and environment. The generated graph shows those transformations. (Link to pdf file: https://drive.google.com/file/d/1zLkHsK9tPLPrvq3OWRqzD0vcxu-xWx4H/view?usp=share_link)



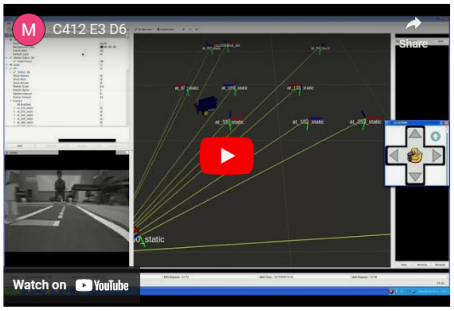
Graph 1 - Screen shot of the Transform Tree Graph (maintains visibility of the nodes)

The next following steps are where everything starts fitting together, now, the next step of this task is to visualize the robot's transforms in RViz. After that, we had to connect the odometry frame and robot frame together. This will then generate a different transform tree graph where the root is the world node instead. This is shown in graph 2. (Link to the pdf file: https://drive.google.com/file/d/1laZk751Q9XA7iW9YL1vXrXH95RSS37hu/view?usp=share_link)

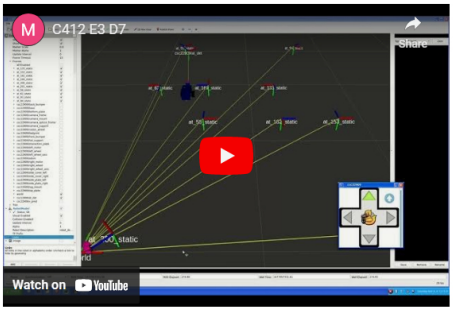


Graph 2 - Transform Tree Graph with the connection between the odometry and robot frame

Now to finalize this section of the lab, my partner Marcus has modified the april tag code that he constructed such that he can then broadcast the transforms of the camera to the closest april being seen by the robot camera. This can be later visualized in RViz with a connecting arc from the robot to the nearest located april tag. Shown in video 5. This part had become a little challenging because to find the right frame to broadcast the transform on since there are multiple camera-related frames to choose from (camera_support, camera_optical_frame, etc.), which can definitely cause confusion without guidance, thankfully Justin was there to help. Finally, we can calculate the transformation of our april tags to the robot wheelbase within the world frame. By applying the transformation to the april tag we can better localize our robot based on that broadcasted april tag. Therefore when we visualize this process in action in RViz and focus on a particular april tag, the robot would "teleport" based within the world frame based on the known location of the april that it is not static in this particular scenario. This can be seen in video 6. The visualization tool RViz has been useful for representing the world and robot frame mentioned in the previous lab. It almost serves as a form of mapping in our case.



Video 5 - RViz visualization of april tag detection with robot, world and



Video 6 - RViz visualization of robot "teleporting" if april tag has been

odometry frames attached

localized.

Questions related to the third task:

Part 3.2

- Where did your odometry seem to drift the most? Why would that be?

It drifts the most when the robot is making turns. It might be caused by:

1. The odometry cannot account for the drifts and inconsistencies of the ground friction at different locations. So the same amount of wheel rotation might result in different turns every time.

2. It appears that the odometry node did not account for the trim parameter, so for robots that had a non-zero trim, the odometry tends to overestimate the amount of distance travelled on one side.

- Did adding the landmarks make it easier to understand where and when the odometry drifted?

Yes, the landmarks served as good reference so that we can compare the relative positions of the odometry and the landmarks nearby to find out how much has the odometry drifted.

Part 3.3

- What is the root/parent frame?

  The parent frame given by our transformed tree graph is represented by the node world and root frame represented in the tree graph as `bot_name/footprint`.

Part 3.4

* what type of joint?

it's continuous. [urdf](https://github.com/duckietown/dt-duckiebot-interface/blob/56a299aa5739e7f03a6b96d3b8dac3a8beca532c/packages/duckiebot_interface/urdf/duckiebot.urdf.xacro)

```
<joint name="${veh}_left_wheel_axis_to_left_wheel"
   type="${'fixed' if model == 'DB18' else 'continuous'}">
<parent link="${veh}/left_wheel_axis" />
<child link="${veh}/left_wheel" />
<axis xyz="0 1 0" />
</joint>
```

Part 3.5

You may notice that the wheel frames rotate when you rotate the wheels, but the frames never move from the origin? Even if you launch your odometry node the duckiebot's frames do not move. Why is that?

Because the odometry is not yet attached to the duckiebot's frames.

1. Using a static transform attach your odometry child frame to the parent frame from the URDF.

- What should the translation and rotation be from the odometry child to robot parent frame? In what situation would you have to use something different?

  In our use case, we can just use a static transform with all zeros for translation and rotation. But when the odometry we have is in a different coordination system, then it becomes necessary to set proper translation and rotation parameters.

- After creating this link generate a new transform tree graph. What is the new root/parent frame for your environment?

  The world frame is now the root.

- Can a frame have two parents? What is your reasoning for this?

  In TF, it is not possible for a frame to have two parents:

  1. The transform tree is a tree structure, i.e., TF expects the frames to form a tree.

  2. In the code, it is not possible to specify more than one parent when constructing the transform object.

- Can an environment have more than one parent/root frame?

  Yes. When we have two (or more) detached transform trees in an environment, then it's possible to have more than one parent/root frame.

Part 3.6

Difficulties: find the right frame to broadcast the tranform on, since there are multiple camera related frames (camera_support, camera_optical_frame, etc.).

- How far off are your detections from the static ground truth.

  It is fairly accurate at the beginning (when the robots hasn't moved yet), but it gets increasingly off as the robot moves.

- What are two factors that could cause this error?

  1. The odometry of the robot has drifted from the ground truth so much, that the broadcasted apriltag estimated locations based on the current robot position are off.

  2. When the robot gets too far (false positives and erroneous pose estimations) or too close (detections from a extreme angle) from the apriltag, the pose estimation returned by the library tends to be inaccurate.

Part 3.7

- Is this a perfect system?

  No. Occasionally it will still gives us very unlikely or outright wrong pose estimations.

- What are the causes for some of the errors?

In addition to the above-mentioned factors, also:

  1. There are delays in between the apriltag detection and the actual position update, so if the robot has moved within the interval, the position update will likely be sending the position estimate to where the robot was a bit ago, instead of where it should be now.

  2. The undistortion and calibration process and many of our measurements (tag size, tag positions and orientations) are not perfect.

- What other approaches could you use to improve localization?

  One think that I can think of is incorporate the linear acceleration and angular velocity measurements from the Inertial Measurement Unit on board (according to the DB21 [datasheet](https://docs.rs-online.com/c57a/A700000007343652.pdf) to help the odometry record information on how much the robot has actually moved.

Looking back on this assignment, we both learned quite a bit, from how to use OpenCV, how april tags work and the different kinds, and how to visualize/detect them, the basics of P.I.D controllers, which are very prominent in several different commercial and scientific settings and how to use and tune them based on the objective at home. And finally, we learned how to do mapping and sensor fusion to be able to localize and "visualize" ourselves within a world frame. This last one has the potential to become super important and serve as a tool for inverse kinematics and other robotic hardware,

such as robotic arms. Despite some of the situations from the previous lab did improve, in my case, efficiency because a bit of time was lost in working on aspects of the lane following node that did not become of any use, which slowed us down a bit. Another thing I found was that despite managing to split the work somewhat evenly. It did feel like Marcus did a bit more work than me. In this part, I hope to improve on the upcoming assignments either by getting together more or finding ways to contribute more to the code. In terms of my partner Marcus, he didn't ask for much help from the TAs in this lab. For example, there was one occurrence where he measured all the april tag coordinates by himself then the TAs later posted those same coordinators. Also, asking for help from the TAs could clarify some ideas or look into alternatives if the solution currently being developed has some issues. Other than that, we are both satisfied with what we have achieved in this lab, and we hope the next ones are just as interesting to work on.

References :

- Transformers: http://wiki.ros.org/tf2
- April tags: https://github.com/AprilRobotics/apriltag/wiki/AprilTag-User-Guide#pose-estimation
- Static frames: http://wiki.ros.org/tf2/Tutorials/Writing%20a%20tf2%20static%20broadcaster%20%28Python%29
- RViz: http://wiki.ros.org/rviz
- OpenCV:
  1. https://docs.opencv.org/3.3.0/dc/dbb/tutorial_py_calibration.html
  2. https://www.tutorialspoint.com/detection-of-a-specific-color-blue-here-using-opencv-with-python
  3. https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html (Others from the OpenCV documentation as well)
- P.I.D controllers:
  1. https://www.youtube.com/watch?v=y3K6FUgrgXw
  2. https://github.com/vazgriz/PID_Controller/blob/master/Assets/Scripts/PID_Controller.cs

Received help from: Justin (TA), Xiao (TA), Marcus (Partner & student)