Individual Lab Report 11

Nathaniel Chapman

Team B: Space Jockey

With Brian Boyle, Ardya Dipta Nandaviri, Songjie Zhong

April 16th, 2014

1 Introduction

Although the project is still behind where we want to be, we have made great strides over the last two weeks, and all of the major software system components are really starting to come together. As demoed, the robot can now warp it's camera image into the world frame, which is the last puzzle piece needed to start integrating our flaw detection code into the system, and the AR tag module is working properly now, which is the last stepping stone necessary to start implementing our localization code.

2 Individual Progress

My first priority for the last two weeks has been to fix the problems that arose during our last progress review. On the robot control side, all of our calibration settings have been moved off of the Arduino and into our ROS configuration. This allows us to update the wiring configuration, and hardware joint limits quickly and easily. One this was done, I refactored the URDF definition of our robot to be coordinate system conventions compliant, so that I could begin debugging our IK engine.

I found that the biggest cause of our problems at the last progress review was the fact our IK engine was producing incorrect solutions. I spent a large portion of my time in trying to clean up our implementation and get it working properly. The biggest issue I encountered was due to the number of joints in our robot. When I initially designed the IK I was working on the idea that the robot has only 7 degrees of freedom, but the free-swivel joint on the center node represents and uncontrolled 8th degree of freedom. However, because the orientation of the robot is uniquely determined by the two feet of the robot attached to the surface, I found I was

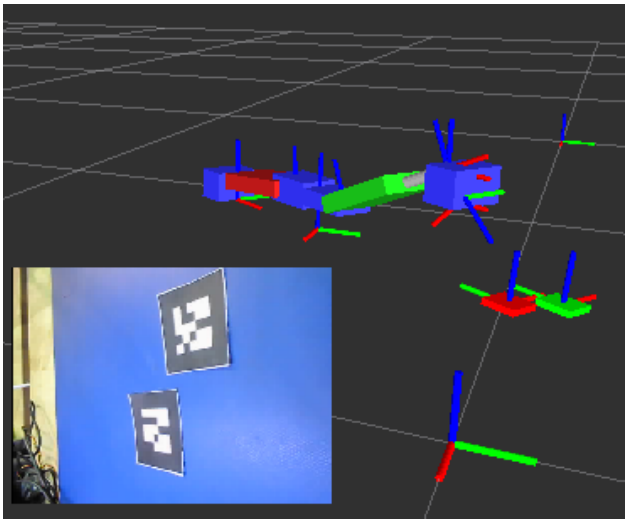

FIGURE 1 - AR TAG LOCATIONS SHOWN IN RVIZ, AND ORIGINAL CAMERA IMAGERY (INSET)

able to easily simplify the IK solution by using only the two outer foot locations relative to the center foot's location in the world rotation frame. I rewrote our IK engine to use these settings as inputs (and support partial solutions for just one foot or the other), and everything is working as expected.

Once our IK was working, I proceeded to fix our minor planner and joint scheduler nodes. I had a great deal of problems getting ROS's tf library to provide the proper transformations due to poor handling of infrequent transform updates. I was able to work around this by adding a "quasi-static" transform re-publisher node, which listens for specially named transforms, and then re-outputs them in the world frame at a higher rate. In the next week, I will be further improving on this node to support weighted averaging of transformations to implement our localization functions.

With the IK and planners working as expected, I turned my attention to merging Songjie and Dipta's code into our main branch. Because much of their work was dependent on many other ros libraries and nodes, I carefully tracked all the project dependencies, and checked the code out into our repository using git submodules, so that all team members are using the same code versions for our project. After that was done, I performed code reviews on both Dipta and Songjie's code, and cherry-picked files from their branches to integrate everything together. As shown in this week's demo, we have AR tag detection and world-frame mapping of the integrated into our system now (Figure 1, above), and transformations of the camera image onto the world frame. A full node graph of our current ROS implementation can be seen in Figure 2 below. For the next demo, we will be integrating Songjie's image comparison code, and my weighted transformation code to properly implement localization and inspection using Dipta's AR tag work.

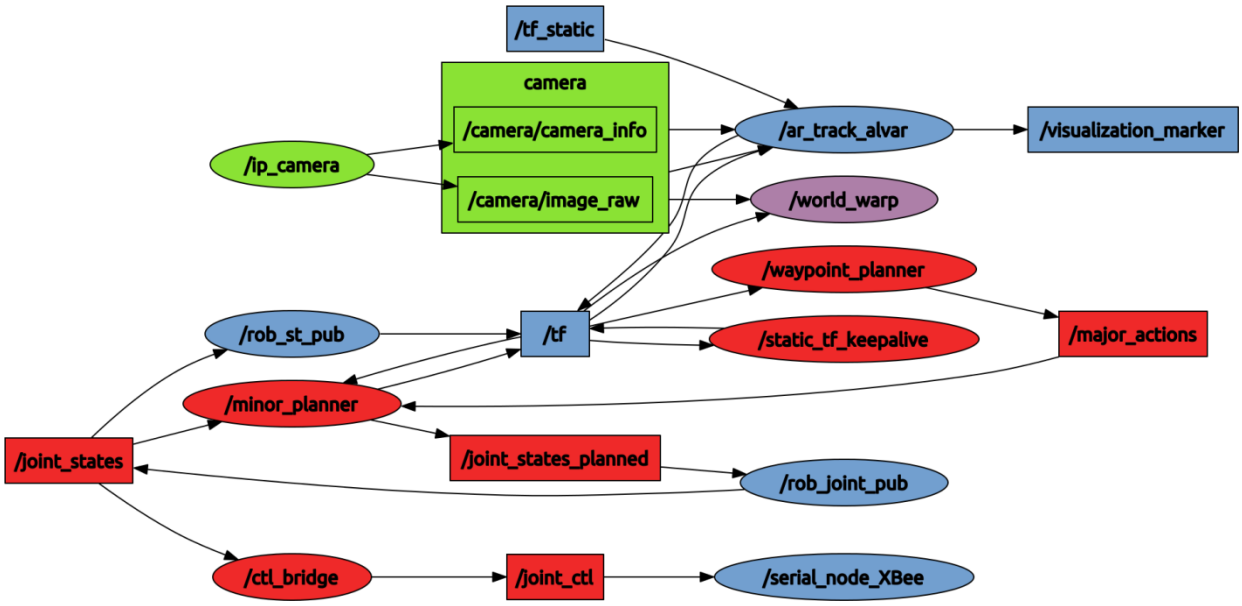

FIGURE 2 - ROSGRAPH OUTPUT OF CURRENT SYSTEM IMPLEMENTATION, SHOWING CAMERA IMPLEMENTATION (GREEN, DIPTA), CV IMAGE WARP NODES (PURPLE, SONGJIE), AND PLANNER PIPELINE (RED, ME), BLUE NODES ARE ROS BUILT-IN MODULES.

3 Challenges / Issues

As mentioned before, most of my implementation difficulties came in implementing the Inverse Kinematics module, although adding urdf parsing and an Rviz testbed made that problem much easier to solve. In addition, I encountered a lot of issues arising out of the time-sensitive nature of ROS's homogenous transform library. There is a new version, tf2 out, but it is poorly documented and difficult to learn. However tf2 does support static transformations natively, so I may look into rebuilding my weighted transform stack based on tf2 in the next week. Right now, I am spinning that module off into a separate code repository, so that I can work on it independently without breaking our main code branch, which is needed for testing and fine-tuning of the attachment mechanism.

4 Efforts by Team Members

During the last two weeks, Brian has primarily been focused on all the physical assembly, testing, and calibration of our robot's frame. He designed and assembled the new (larger, to resist moments) foot segments and camera mount, and did all of the servo calibrations, updating the robot's XML description in ROS to keep our simulations in sync with reality. Songjie has been working hard to get the camera image data properly warped into the world frame, so that he can begin to do our comparison and flaw detection code integrated with the project. This was demoed yesterday. Dipta has been debugged his wifi camera driver, calibrated the camera lens parameters, and merged his code with ROS' AR tag package and the system as a whole.

5 Future Plans

For the SVE demo next week, I plan to focus on getting our localization code working, and work the bugs out of the joint scheduler to prevent detachment from the vertical surface. Our current plan is to try and get everything working on the wall, but if we are still having difficulties with the attachment mechanism, we may do our test procedure on a flat surface, so we can verify all of our other performance characteristics in lieu of the attachment mechanism. I will also be working closely with Songjie to get all of his flaw detection code integrated with ROS, so that we can complete all of our desired inspection tasks as planned.

For the SVE redux in two weeks, we will focus on getting the attachment mechanism working as expected, doing a full code review, and trying to integrate everything together in one GUI window. The new user friendly GUI will allow the user to set waypoints by clicking (move to location), right clicking (view point), or clicking and dragging (inspect region). The updated world map from the inspection process will be shown in the GUI, and all flaws will be marked in the same window. Our goal is to get the entire system working cleanly and simply, without any complex user training necessary.