Computer Vision Enabled Localization and Deployment of Mobile Robot Networks

Katherine Liu
University of California, San Diego
Spring 2014

0 ABSTRACT

This paper summaries my individual efforts to implement a computer-vision enabled state estimation framework. That past quarter, I have focused mainly on increasing the robustness of the system and testing the robotic framework on a larger number of agents. The integration of a new library significantly decreased the number of codes and complexity, while increasing tracking efficiency.

1 BIG PICTURE

This section will cover both the overall objectives of the project, as well the objectives of my own sub-project.

1.1 Objective and Overall Goals

Realization of control algorithms in the physical world necessitates a readily deployable system. The Cortes and Martinez labs at UCSD currently possess ten TurtleBot platforms to be integrated into a multi-agent robotic network. The open source ROS has been chosen as the software platform to enable the network structure, relying on a publisher-subscriber model to allow for inter-robot communication and coordination.

The objective of the ROS TurtleBot project is to provide a stable system to which distributed control algorithms can be readily deployed for testing and validation on hardware. There are a plethora of interesting algorithms that could potentially be deployed using the TurtleBot network, such as cooperative task completion. Of particular interest to the group is cooperative simultaneous localization and mapping (SLAM) and cooperative coverage control.

1.2 Localization of Mobile Robots

Allowing individual agents in a network of autonomous robots to access their location information is a valuable tool in deploying many control algorithms. Discrete sampling of agent location can be used to generate velocity estimates as well as heading information. In a GPS denied environment, computer vision algorithms serve as a means by which to extract location information from images collected by a camera. This information can be leveraged as a ground truth by which to benchmark other control algorithms, such as SLAM.

To this end, we desire a computer vision enabled localizing node capable of integrating with the Linux based ROS that leverages the open source C++ library, OpenCV. The node should be capable of delivering both location information in the form of Cartesian coordinate and heading information describing the general direction of the robot. Furthermore, the localization scheme should be scalable, so that as the number of agents in the network increases, the capability is still supported.

2 PERSONAL CONTRIBUTIONS

My contributions this quarter are continuations of my work of the previous quarter. For details regarding prior work, please refer to my Winter 2014 report.

This quarter I spent some time building the group infrastructure and productivity tools. The group now has a general email address at ucsd.distributed.ros.project@gmail.com, as well as a website. The website is currently a work in progress, and will be updated with media in the coming weeks. I have also begun a Sway interface for the project as an easy presentation-style link. In terms of productivity, I have also begun a GitHub for the group. GitHub provides a few advantages over SVN, namely in the support of branches. This allows multiple people to work on the same pieces of code while preserving “safe”, validated code, and also plays nicely with the ROS catkin workspace. GitHub also has a wiki interface that can be used for user manuals and guides. Portions of this wiki are included in the Appendix of this report.

This quarter I also successfully replaced the computer vision node with the ArUco library. More information can be found here. In summary, the integration of this extremely efficient library results in a substantial reduction in lines of code, better results in tracking, and tolerance to changes in the environment.

In terms of deployment, this quarter I have made much progress in altering the existing framework to allow for the more efficient deployment of robots. A deployment of six robots has been achieved, and a demo integrating the teleop functionality has also been tested.

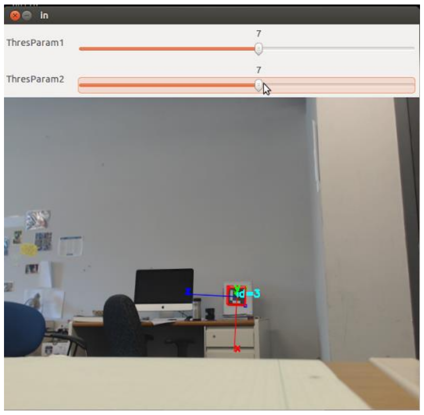
3 PRELIMINARIES

In this section, several basic topics crucial for multi-agent deployment are discussed.

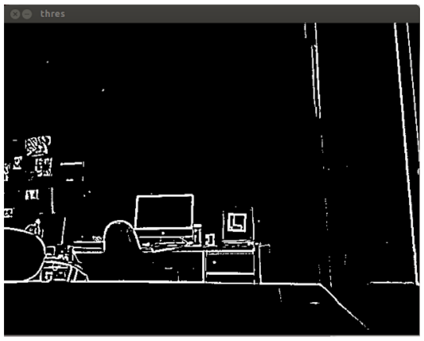
3.1 Aruco Library

In the paper, *“Automatic generation and detection of highly reliable fiducial markers under occlusion”*, the authors from Cordoba University outline the robust tracking of markers not dissimilar to QR codes. For practical purposes, there are several benefits for our use of this library, including:

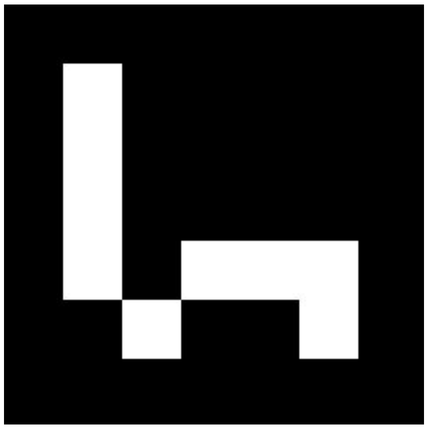
- Black and white are more easily detected in variable lighting environments
- Corner detection is fairly robust
- The ArUco markers have less embedded bits of information than a QR code, making them computationally easier to identify
- The markers contain information, specifically identification numbers
- With knowledge of camera intrinsic, it is possible to extrapolate the orientation of the marker with respect to the camera.



Example of ArUco interface



Thresholded image

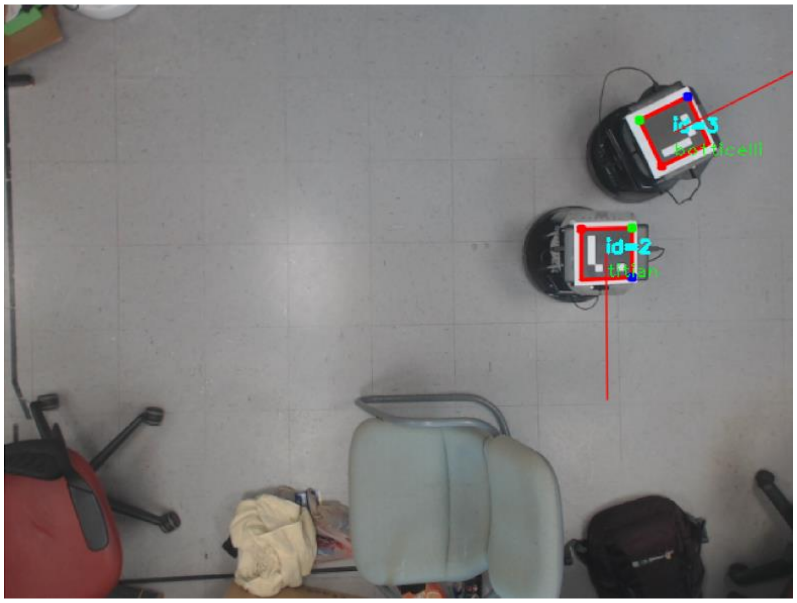

Example marker (ID=6)

4 METHODOLOGY

In this section, each development in regards to robot tracking and deployment are explored in greater depth. For methodology and more detail on the actual tracking algorithm, refer to my Spring 2014 report. Code can be found in the GitHub repository.

4.1 Implementation of Aruco Library

Due to the modular nature of the code infrastructure, the implementation of the Aruco library was less intensive than building from the group up. The library is installed on the Mencia and Calipso computers.



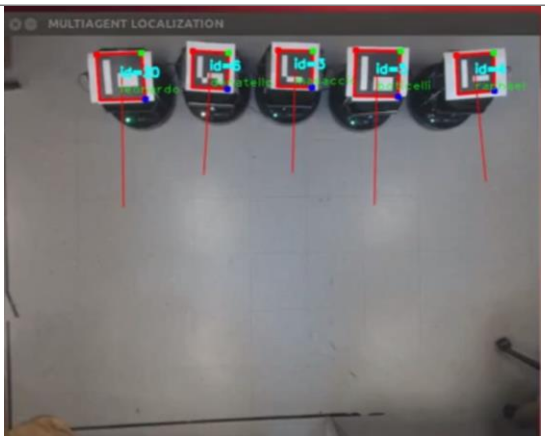
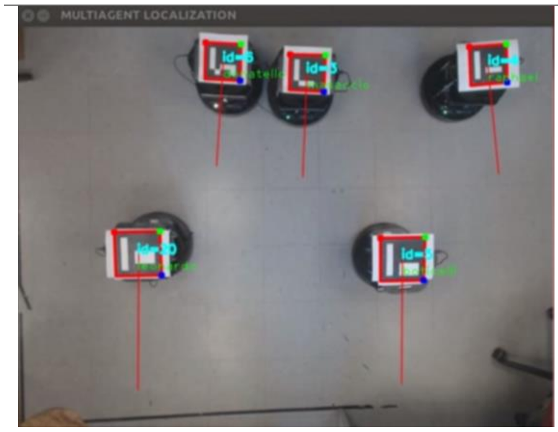
ArUco integration

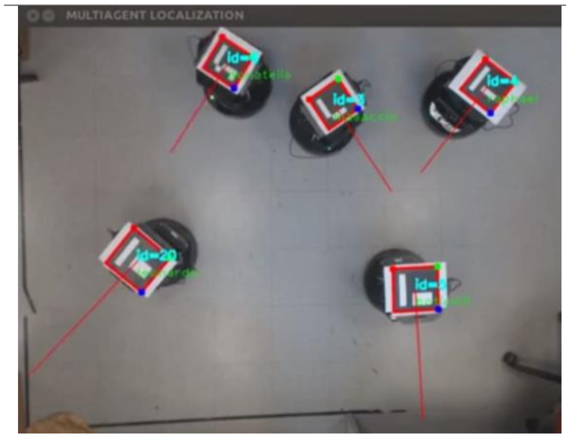
4.2 Multi-agent Deployment

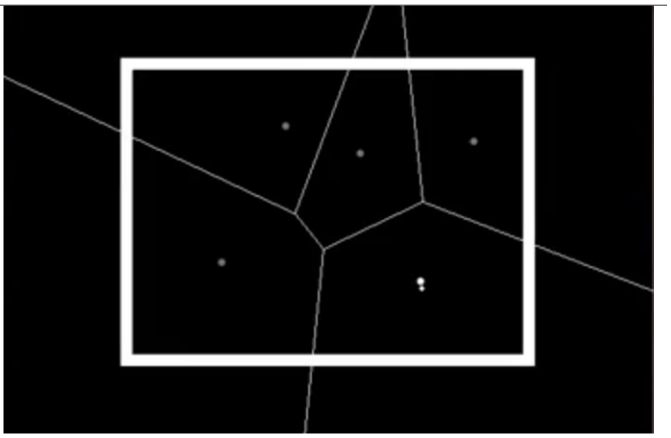
Several multi-agent deployments have been achieved this quarter. Modifications to the existing system included map scaling improvement and turning off visualization of the Voronoi cells for each robot. As each robot individually calculates Voronoi cells and attempts to autonomously decide its goal position, porting each display window over the Wi-Fi proved to consume too much bandwidth, and resulted in high dropout rates. Only attempting to visualize via one robot improved dropout rates, but this is still an area that must be improved.

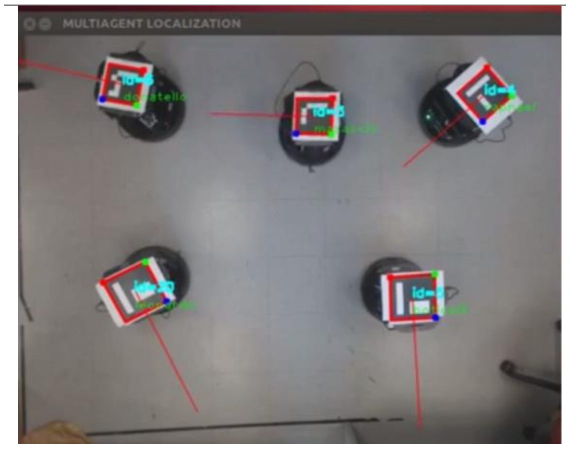
4.2.1 Six TurtleBots

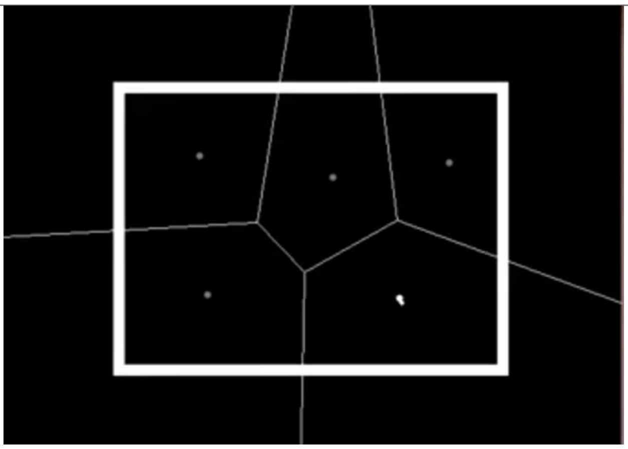
Below is an example of a six-robot deployment with increasing time.

Camera feed	**Vornoi cell visualization**
	No data available
	No data available



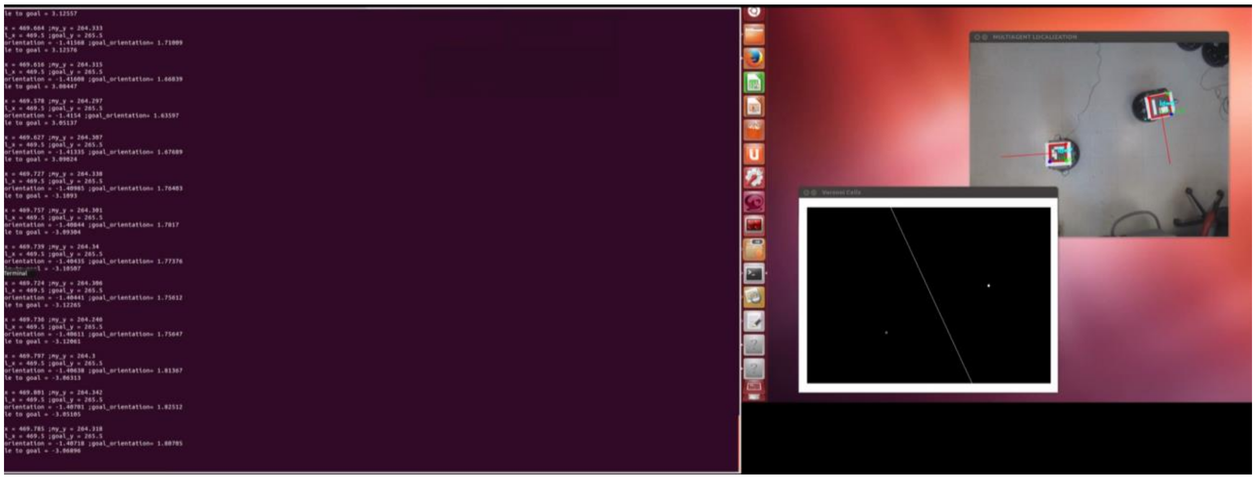






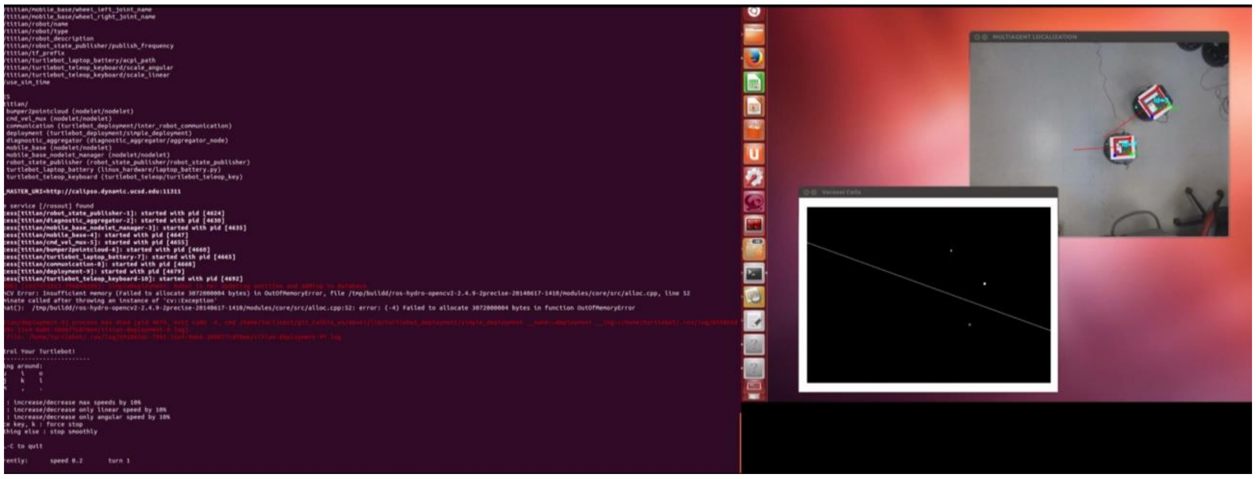
4.3 Teleop Integration

Teleop integration is a useful demonstration tool that is currently in testing. Essentially, the teleop integration allows a user to manually drive around one robot in the system, while the other robots will attempt to achieve coverage control. The modifications involve creating a mode for a robot to still report its position to the /all_positions topic, but also be controlled by the keyboard.



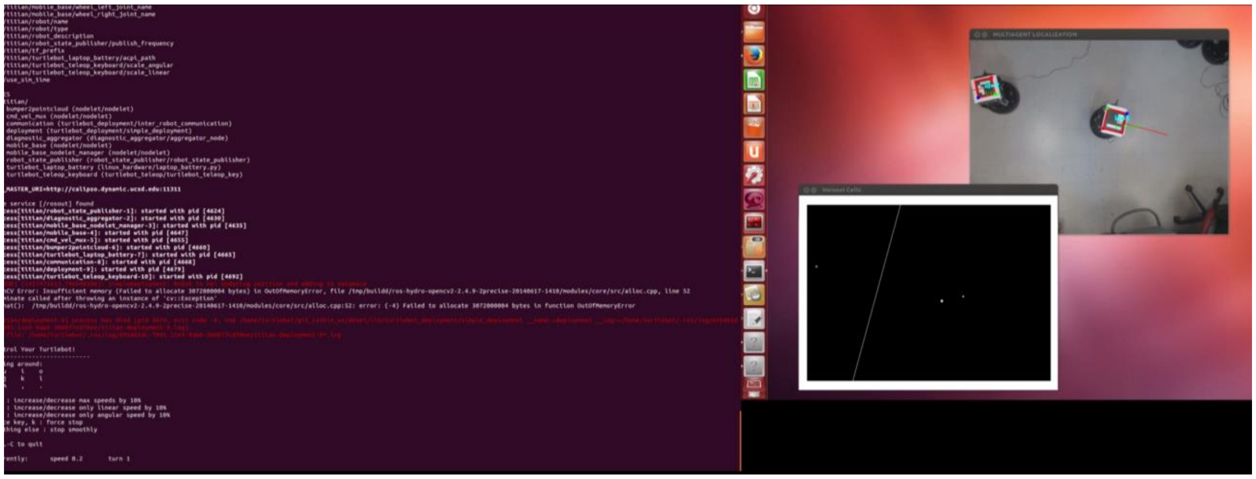
```
le to goal = 3.12557

x = 469.664 ;my_y = 264.333
l_x = 469.5 ;goal_y = 265.5
orientation = -1.41568 ;goal_orientation= 1.71009
le to goal = 3.12576

x = 469.616 ;my_y = 264.315
l_x = 469.5 ;goal_y = 265.5
orientation = -1.41608 ;goal_orientation= 1.66839
le to goal = 3.08447

x = 469.578 ;my_y = 264.297
l_x = 469.5 ;goal_y = 265.5
orientation = -1.4154 ;goal_orientation= 1.63597
le to goal = 3.05137

x = 469.627 ;my_y = 264.307
l_x = 469.5 ;goal_y = 265.5
orientation = -1.41335 ;goal_orientation= 1.67689
le to goal = 3.09024

x = 469.727 ;my_y = 264.338
l_x = 469.5 ;goal_y = 265.5
orientation = -1.40985 ;goal_orientation= 1.76403
le to goal = -3.1093

x = 469.757 ;my_y = 264.301
l_x = 469.5 ;goal_y = 265.5
orientation = -1.40844 ;goal_orientation= 1.7817
le to goal = -3.09304

x = 469.739 ;my_y = 264.34
l_x = 469.5 ;goal_y = 265.5
orientation = -1.40435 ;goal_orientation= 1.77376
le to goal = -3.10507
Terminal
x = 469.724 ;my_y = 264.306
l_x = 469.5 ;goal_y = 265.5
orientation = -1.40441 ;goal_orientation= 1.75612
le to goal = -3.12265

x = 469.736 ;my_y = 264.246
l_x = 469.5 ;goal_y = 265.5
orientation = -1.40611 ;goal_orientation= 1.75647
le to goal = -3.12061

x = 469.797 ;my_y = 264.3
l_x = 469.5 ;goal_y = 265.5
orientation = -1.40638 ;goal_orientation= 1.81367
le to goal = -3.06313

x = 469.801 ;my_y = 264.342
l_x = 469.5 ;goal_y = 265.5
orientation = -1.40701 ;goal_orientation= 1.82512
le to goal = -3.05105

x = 469.785 ;my_y = 264.318
l_x = 469.5 ;goal_y = 265.5
orientation = -1.40718 ;goal_orientation= 1.80705
le to goal = -3.06896
```

Teleop integration start-up



```
/tittan/mobile_base/wheel_left_joint_name
/tittan/mobile_base/wheel_right_joint_name
/tittan/robot/name
/tittan/robot/type
/tittan/robot_description
/tittan/robot_state_publisher/publish_frequency
/tittan/tf_prefix
/tittan/turtlebot_laptop_battery/acpi_path
/tittan/turtlebot_teleop_keyboard/scale_angular
/tittan/turtlebot_teleop_keyboard/scale_linear
/use_sim_time

ES
titan/
 bumper2pointcloud (nodelet/nodelet)
 cmd_vel_mux (nodelet/nodelet)
 communication (turtlebot_deployment/inter_robot_communication)
 deployment (turtlebot_deployment/simple_deployment)
 diagnostic_aggregator (diagnostic_aggregator/aggregator_node)
 mobile_base (nodelet/nodelet)
 mobile_base_nodelet_manager (nodelet/nodelet)
 robot_state_publisher (robot_state_publisher/robot_state_publisher)
 turtlebot_laptop_battery (linux_hardware/laptop_battery.py)
 turtlebot_teleop_keyboard (turtlebot_teleop/turtlebot_teleop_key)

_MASTER_URI=http://calipso.dynamic.ucsd.edu:11311

e service [/rosout] found
cess[tittan/robot_state_publisher-1]: started with pid [4624]
cess[tittan/diagnostic_aggregator-2]: started with pid [4630]
cess[tittan/mobile_base_nodelet_manager-3]: started with pid [4635]
cess[tittan/mobile_base-4]: started with pid [4647]
cess[tittan/cmd_vel_mux-5]: started with pid [4655]
cess[tittan/bumper2pointcloud-6]: started with pid [4660]
cess[tittan/turtlebot_laptop_battery-7]: started with pid [4665]
cess[tittan/communication-8]: started with pid [4668]
cess[tittan/deployment-9]: started with pid [4679]
cess[tittan/turtlebot_teleop_keyboard-10]: started with pid [4692]
[illegible]
nCV Error: Insufficient memory (Failed to allocate 3072000004 bytes) in OutOfMemoryError, file /tmp/buildd/ros-hydro-opencv2-2.4.9-2precise-20140617-1410/modules/core/src/alloc.cpp, line 52
minate called after throwing an instance of 'cv::Exception'
hat():  /tmp/buildd/ros-hydro-opencv2-2.4.9-2precise-20140617-1410/modules/core/src/alloc.cpp:52: error: (-4) Failed to allocate 3072000004 bytes in function OutOfMemoryError

[illegible]

trol Your Turtlebot!
---------------------------
ing around:
u    i    o
j    k    l
m    ,    .

 : increase/decrease max speeds by 10%
 : increase/decrease only linear speed by 10%
 : increase/decrease only angular speed by 10%
ce key, k : force stop
thing else : stop smoothly

L-C to quit

rently:      speed 0.2      turn 1
```

Teleop integration t1

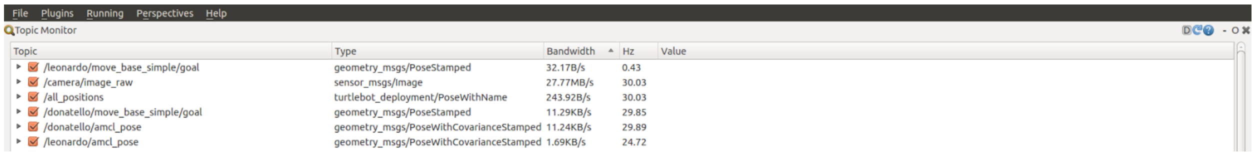


```
/tittan/mobile_base/wheel_left_joint_name
/tittan/mobile_base/wheel_right_joint_name
/tittan/robot/name
/tittan/robot/type
/tittan/robot_description
/tittan/robot_state_publisher/publish_frequency
/tittan/tf_prefix
/tittan/turtlebot_laptop_battery/acpi_path
/tittan/turtlebot_teleop_keyboard/scale_angular
/tittan/turtlebot_teleop_keyboard/scale_linear
/use_sim_time

ES
titan/
 bumper2pointcloud (nodelet/nodelet)
 cmd_vel_mux (nodelet/nodelet)
 communication (turtlebot_deployment/inter_robot_communication)
 deployment (turtlebot_deployment/simple_deployment)
 diagnostic_aggregator (diagnostic_aggregator/aggregator_node)
 mobile_base (nodelet/nodelet)
 mobile_base_nodelet_manager (nodelet/nodelet)
 robot_state_publisher (robot_state_publisher/robot_state_publisher)
 turtlebot_laptop_battery (linux_hardware/laptop_battery.py)
 turtlebot_teleop_keyboard (turtlebot_teleop/turtlebot_teleop_key)

_MASTER_URI=http://calipso.dynamic.ucsd.edu:11311

e service [/rosout] found
cess[tittan/robot_state_publisher-1]: started with pid [4624]
cess[tittan/diagnostic_aggregator-2]: started with pid [4630]
cess[tittan/mobile_base_nodelet_manager-3]: started with pid [4635]
cess[tittan/mobile_base-4]: started with pid [4647]
cess[tittan/cmd_vel_mux-5]: started with pid [4655]
cess[tittan/bumper2pointcloud-6]: started with pid [4660]
cess[tittan/turtlebot_laptop_battery-7]: started with pid [4665]
cess[tittan/communication-8]: started with pid [4668]
cess[tittan/deployment-9]: started with pid [4679]
cess[tittan/turtlebot_teleop_keyboard-10]: started with pid [4692]
[illegible]
nCV Error: Insufficient memory (Failed to allocate 3072000004 bytes) in OutOfMemoryError, file /tmp/buildd/ros-hydro-opencv2-2.4.9-2precise-20140617-1410/modules/core/src/alloc.cpp, line 52
minate called after throwing an instance of 'cv::Exception'
hat():  /tmp/buildd/ros-hydro-opencv2-2.4.9-2precise-20140617-1410/modules/core/src/alloc.cpp:52: error: (-4) Failed to allocate 3072000004 bytes in function OutOfMemoryError

[illegible]

trol Your Turtlebot!
---------------------------
ing around:
u    i    o
j    k    l
m    ,    .

 : increase/decrease max speeds by 10%
 : increase/decrease only linear speed by 10%
 : increase/decrease only angular speed by 10%
ce key, k : force stop
thing else : stop smoothly

L-C to quit

rently:      speed 0.2      turn 1
```

Teleop integration t2

4.4 System studies

This quarter, I conducted several system studies. The screenshot below shows that for two robots the camera image is delivered at about 30 fps as expected, and the pose as calculated by the ArUco library is also at about 30 Hz. However, the Voronoi calculation and subsequent publishing of the goal happens every two seconds, which is most likely a source of jerky deployment. To improve this, I suggest looking into more efficient geometric calculation libraries specifically optimized for this application.



Topic	Type	Bandwidth	Hz	Value
/leonardo/move_base_simple/goal	geometry_msgs/PoseStamped	32.17B/s	0.43	
/camera/image_raw	sensor_msgs/Image	27.77MB/s	30.03	
/all_positions	turtlebot_deployment/PoseWithName	243.92B/s	30.03	
/donatello/move_base_simple/goal	geometry_msgs/PoseStamped	11.29KB/s	29.85	
/donatello/amcl_pose	geometry_msgs/PoseWithCovarianceStamped	11.24KB/s	29.89	
/leonardo/amcl_pose	geometry_msgs/PoseWithCovarianceStamped	1.69KB/s	24.72	

Bandwidth and frequency study of two robots

5 CONCLUSIONS

This section includes general troubleshooting tips for working with ROS, and a suggested path forward.

5.1 Troubleshooting/Tips

Trouble-shooting Kobuki base problems

> Try running top. If you see anything on the turtlebot that was created by the "ros" user, it's probably residual processes that are hogging resources or something.
>
> ```
> sudo killall -u ros
> ```
>
> If you don't see this, which I didn't for one robot, try a reboot, and then check again. If there are ros processes, kill them. This is a hacky solution, and may not work for all the robots but so far it's worked on at least two.

5.2 Path forward

It will be useful for the group to integrate the navigation stack for more efficient and aesthetic deployments. The ROS navigation stack also has the benefit of obstacle avoidance via the Kinect. Efforts should be taken to improve the goal position transmission rate. Possible solutions include using more efficient Vornoi-cell calculate libraries, or scaling down the map. Finally, next quarter I will also be working on combining the data from two different webcams to increase the area surveyed.

Beginner’s GitHub User Guide

GitHub is a more sophisticated version of SVN that allows for branching. There are plenty of tutorials that you can find online. Basically, with branches, there's not only version control, but some safety in that any number of people can work on the code at the same time and there are protocols for recombining things into the Master branch. The general idea is that the Master branch should be a perfectly working version of code. Any new functionality, however small of a "risk" it seems to pose should be developed in a separate, personal branch. When you feel it's good enough to merge back to master, you then ask a peer to review your code before it is officially allowed into the pristine Master branch.

How to set up this repo for ROS

Create a folder

It will be convenient to put this folder in your home directory.

```
$ mkdir git_catkin_ws
$ mkdir git_catkin_ws/src
```

Initialize the catkin workspace

```
$ cd ~/git_catkin_ws/src
$ catkin_init_workspace
$ cd ..
$ catkin_make
```

You should now see /build and /devel folders.

Clone this repo

```
$ cd src
$ git clone https://github.com/kat-liu/ucsd_ros_project
```

Here you will be asked to enter your git user information. You'll need to have registered on GitHub first, and then requested permission from the admin. Send an email to kyl037(at)ucsd.edu if you haven't requested permission yet. Don't forget to actually compile this code by changing directory to git_catkin_ws again and running catkin_make.

Reroute .bashrc

```
$ gedit ~/.bashrc
```

Look to the bottom of the file. You'll probably see a line like this:

source ~/catkin_ws/devel/setup.bash

Change this to point to the new git_catkin_ws by making it

source ~/git_catkin_ws/devel/setup.bash

Save and close the file. Source the file.

```
$ source ~/.bashrc
```

Workflow

This section gives an example of a workflow. As always, Google is a great resource for git commands. We'll pretend that I want to add a functionality where the Turtlebots don't make noises when they boot up.

Pull the most recent code

cd to ucsd_ros_project in your catkin workspace

```
$ git checkout master
$ git pull origin master
```

These commands change you to the master branch, and then pull the most recent code from master. This is a pretty crucial step, as the code in Master is constantly being updated, and you usually want to start developing from the cleanest state of code. In cases where you might want to branch off another branch, you should just run this command

```
$ git checkout some_other_branch
```
etc.

Make your own branch

```
$ git checkout -b turn_off_sound
```
Checkout a new branch called "turn_off_sound"

Make the code changes you desire

Code as normal. When you've fixed something, or are at a good checkpoint, run the following command to save it

```
$ git commit -am "This commit fixes the stdio bug"
```

At big checkmarks, push your code to GitHub

Until you run the following command, your code only exists on the local machine

```
$ git push origin turn_off_sound
```

Pull request time

Once the code works, is tested, and you're ready to merge your branch back to master, you'll have to submit a pull request. Check out this link for more on that. Basically, someone has to review your code and then accept. This step has to be done through the web interface. If you've been asked to review code, there are a few criteria to ask yourself:

- Is this code understandable? It is appropriately commented?
- Does this code work?

GitHub provides a lot of functionality that allows users to review code and request changes, much like a Google Drive editing function- with comments and conversations.

Usually, it's also good practice to merge the master code into your own branch and resolve any conflicts before you move your code into master.

/home/turtlebot/git_catkin_ws/src/ucsd_ros_project/localization-mounted-camera/turtlebot_camera_localization/src/main.cpp:13:25: fatal error: aruco/aruco.h: No such file or directory compilation terminated.

Utilizing Branch Functionality

Don't just edit things on Master. Keep this branch safe by making changes on a different branch. Anyone can access your branch with a simple

```
$ git checkout some_other_branch
```

If you don't see a branch you're looking for, try

```
$ git fetch
```

and then the checkout command again. It's important to note that you can't switch branches unless you've commited on your own branch, so you may have to commit whatever you're working on before switching. /home/turtlebot/git_catkin_ws/src/ucsd_ros_project/localization-mounted-camera/turtlebot_camera_localization/src/main.cpp:13:25: fatal error: aruco/aruco.h: No such file or directory compilation terminated.

To check what branch you're currently on, try

```
$ git branch
```

To get the most recent version of your branch, you'll need to run this command

```
$ git pull origin my_branch_name
```

Things to remember

Just because the code has switched DOES NOT mean that the code you run is the "new" stuff. When you call launch files, you're really calling executables- and these aren't made unless you run `catkin_make` from inside the git_catkin_ws folder. If you see an error like this on the turtlebot:

```
/home/turtlebot/git_catkin_ws/src/ucsd_ros_project/localization-mounted-camera/turtlebot_camera_localization/src/main.cpp:13:25: fatal error: aruco/aruco.h: No such file or directory compilation terminated.
```

don't worry. The Aruco library is not installed on the turtlebots.

Coverage Control Demo

These are the instructions to launch a coverage control demo on multiple robots.

How to launch

On the Master Computer

First terminal - roscore

```
$ roscore
```

Second terminal - Map server

```
$ cd git_catkin_ws/ucsd_ros_project/deployment/turtlebot_deployment/launch
$ roslaunch roslaunch map_server.launch
```

On each Turtlebot - Deployment

These steps assume you've ssh-ed into your Turtlebot of choice, and that the code is all up-to-date. The following steps are for Boticelli

```
$ cd git_catkin_ws/ucsd_ros_project/deployment/turtlebot_deployment/launch
$ roslaunch deploy_robot.launch show_cells_:=true
```

Notes

Note that having the visualization on for many robots slows down the network appreciably, so the robots default to not forwarding the information. If you would like to see the cells, only include the show_cells_ variable assignment on that robots.

See a video here:https://drive.google.com/file/d/0B2D4WxCdUW5VMjZTcG9xLXNoLTQ/view?usp=sharing