Localization and Deployment via Stereo Vision of Mobile Robot Networks

Katherine Liu
University of California, San Diego
Winter 2015

0 ABSTRACT

This paper summaries my individual efforts to implement a computer-vision enabled state estimation framework. That past quarter, I have focused on beginning the framework for extending the existing localization scheme to accommodate two cameras. Also, I have worked on various smaller projects, including media, and designing an interactive demonstration.

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.

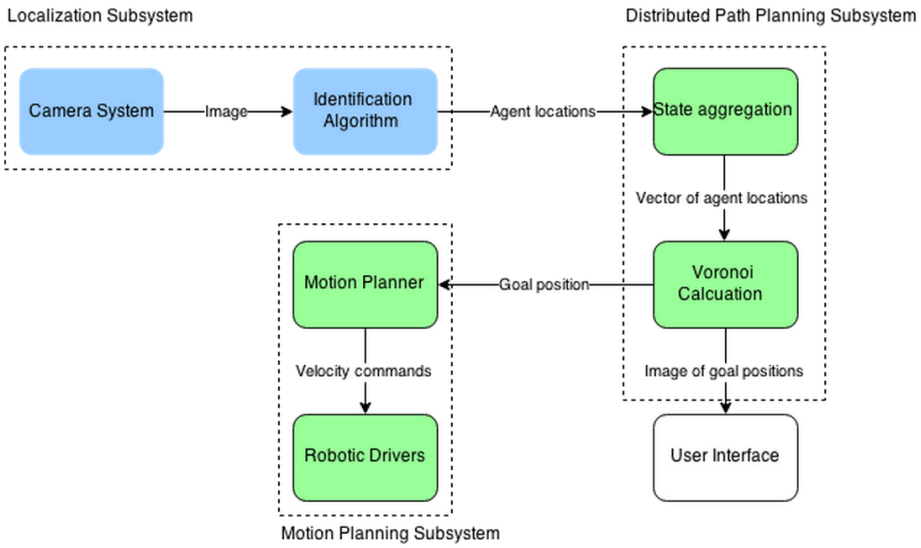

Figure 1: High level system diagram

1.2 Increased Capability Localization of Mobile Agents

In previous quarters, a computer vision enabled system that allows individual agents in a network of autonomous robots to access their location information was researched, designed, and implemented. This allowed for robots to have feedback about their positions, and was used as an external state-estimation framework. However, the use of one camera limits the floor space that can be used for deployment. This work falls within the Localization Subsystem in Figure 1.

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 previous reports. This quarter I worked more on the Sway that was started last quarter. I also started researching how to extend the existing camera framework to two cameras, and begun implementation. One step in this process was to construct a basic visualization node that would be used in lieu of image stitching, and this project is near its final stages. I also integrated a wireless XBOX 360 controller into the system.

3 PRELIMINARIES

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

3.1 Stereo Vision

Stereo vision describes the use of two ocular sensors to determine visual information. Many animals, such as humans, use stereo vision for the encoded depth information.

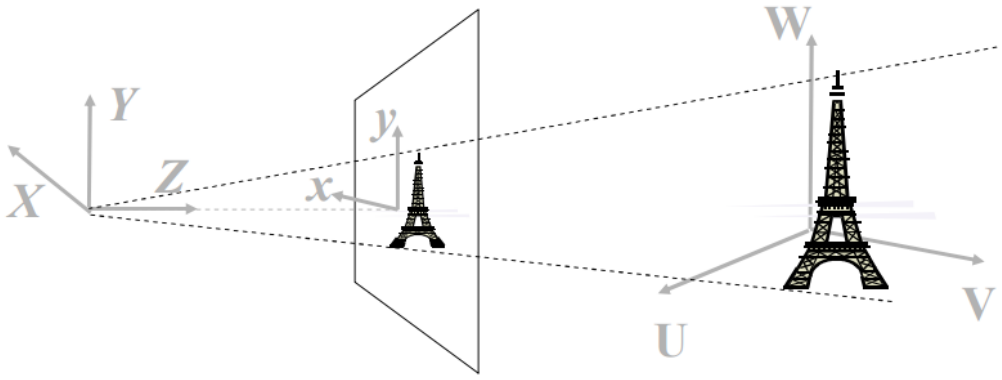

Figure 2[1]: Project from 3D coordinates to image coordinates

[1] Collins, Robert, and Penn State Cse486. Lecture 12: Camera Projection (n.d.): n. pag. Web. <http://www.cse.psu.edu/~rcollins/CSE486/lecture12.pdf>.

The OpenCV library provides documentation on its stereo vision calibration function. Figure 3 is taken from the documentation and describes the relationship between the real world coordinates and the image coordinates.

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

where:

- (X, Y, Z) are the coordinates of a 3D point in the world coordinate space
- (u, v) are the coordinates of the projection point in pixels
- A is a camera matrix, or a matrix of intrinsic parameters
- (cx, cy) is a principal point that is usually at the image center
- fx, fy are the focal lengths expressed in pixel units.

Figure 3[2]: Transformation matrices in stereo vision

Furthermore, there exists some "essential matrix", which describes the transform between the two cameras such that

$$\boldsymbol{E} = \boldsymbol{R}[\boldsymbol{t}]$$

$$\widetilde{x'} = \boldsymbol{R}(\tilde{x} - \boldsymbol{t}) \quad (1)$$

where R is a 3x3 rotation matrix, and t is a 3-dimensional translation vector.

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 Fall 2014 report. Code can be found in the GitHub repository.

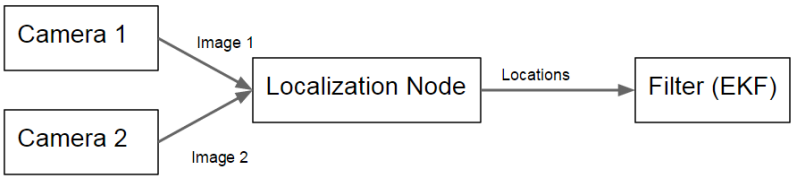

Figure 4: High level algorithmic approach to stereo localization

[2] "Camera Calibration and 3D Reconstruction." Camera Calibration and 3D Reconstruction — OpenCV 2.4.11.0 Documentation. OpenCV, n.d. Web. 20 Mar. 2015. <http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html>.

There are several technical challenges in implementing two cameras to localize the robots. These are summarized in the following table:

Challenge	Approach
Determine the relationship between what is seen in one camera and what is seen by another	Calibration
Difficulty in syncing the camera feeds	Filtering

The calibration can be completed using OpenCV and built in ROS features, although custom calibration files will have to be constructed. The filtering represents an effort by graduate student in the group, Aaron Ma, to write an Extended Kalman Filter. The basic scheme is as follows: the localization node is modified to listen to the topics of both cameras. When any image is received, the algorithm checks to see if it was received by the primary or secondary camera. This determines which subsequent localization function to call. The primary localization function is very similar to that of previous implementations of the system. The secondary localization function applies the transform described in **3.1** to extrapolate the location in the coordinate frame of the primary camera. This in theory allows for all the information to be published in the same coordinate frame.

4.1 Calibration of Stereo Cameras

The calibration of stereo cameras depends on the uvc_cam stereo vision node, which has been moved into the catkin_workspace on the GitHub. The end goal is to use Equation 1 to find the transform from the second camera to the first camera. The user can extract the essential matrix information from the calibration process.

4.1.1 Code

To calibrate, run `roscore` and the `uvc_cam stereo_vision` node. Follow the onscreen directions.

```
roslaunch ~/git_catkin_ws/src/ucsd_ros_project/localization-mounted-
camera/uvc_camera/launch/stereo_node.launch

rosrun camera_calibration cameracalibrator.py --size 8x6 --square 0.108
right:=/right/image_raw left:=/left/image_raw right_camera:=/right
left_camera:=/left
```

4.1.2 Calibration Images

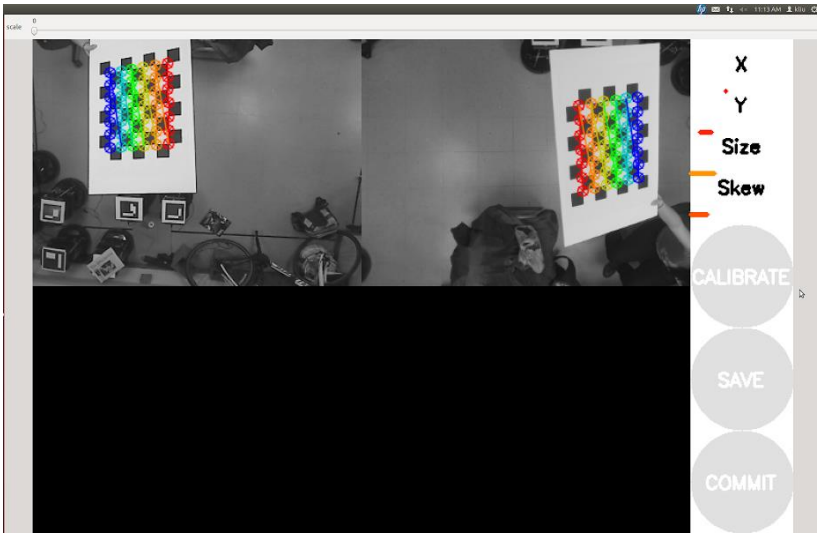

Figure 5: Calibration function during calibration

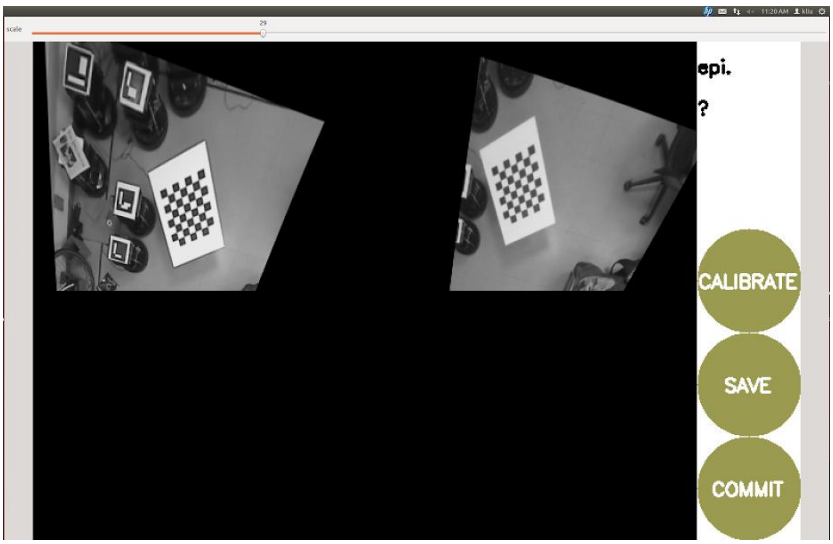

Figure 6: After calibration and rectification

4.2 Visualization

The visualization node exists within the localization node. A visualization node is required because the cameras are not synced and therefore the prospect of stitching the images is technically difficult. While the algorithm could work without visual feedback to the user, such a feature is undeniably useful for debugging and for demonstrations. Therefore, while building the stereo localization, a basic global visualization feature is also being constructed.

The main function is changed such that it reads parameters in the launch file to determine how many cameras the user wants to deploy. The following code snippet demonstrates how this information is handled. Note that checking to see if the custom-made stereo params file has been opened is key.

```
if (camera_number > 1)
	{
		ROS_ERROR("Camera number is larger than one");
		sub1 = it.subscribe("/left/image_raw", 1, imageCallbackLeft);
		sub2 = it.subscribe("/right/image_raw", 1, imageCallbackRight);
		cv::FileStorage fs2(stereo_params, cv::FileStorage::READ);
		if (!fs2.isOpened()) {
			ROS_ERROR("Couldn't open stereo params file. Please restart");
			cout << "Tried to open " << stereo_params << endl;
		}
		fs2["R"] >> R;
		fs2["T"] >> T;
		cout << R << endl;
		cout << T << endl;
	}
	else {
		sub = it.subscribe("camera/image_raw", 1, imageCallbackLeft);
	}
```

The following is a code snippet of the function called when a new image is published to the left camera:

```
//This function is called everytime a new image is published to the left camera
void imageCallbackLeft(const sensor_msgs::ImageConstPtr& original_image)
{
    ROS_ERROR("image callback left");
    current_time = ros::Time::now();

    //Convert from the ROS image message to a CvImage suitable for working with OpenCV
for processing
    cv_bridge::CvImagePtr cv_ptr;
    try
    {
        cv_ptr = cv_bridge::toCvCopy(original_image, enc::BGR8);
    }
    catch (cv_bridge::Exception& e)
    {
        //if there is an error during conversion, display it
        ROS_ERROR("tutorialROSOpenCV::main.cpp::cv_bridge exception: %s", e.what());
        return;
    }
    //Call to Aruco to identify markers
    locate_markers(cv_ptr, 1);
}
```

The `locate_markers` function simply uses the algorithm described in previous reports, and applies a transform if necessary. Both the left and right cameras publish to the same localization topics. Then they show a marker and label giving the position of the turtlebot in a common reference frame, as is seen in **4.2.1.**

4.2.1 Visualization Testing

There are still a few bugs in the visualization. Firstly, the transform is not yet properly applied. However, the overall framework of showing the information from both cameras has been verified to work. This is shown in the following figures. Note that the "locations" found for both camera frames show up on the same window.

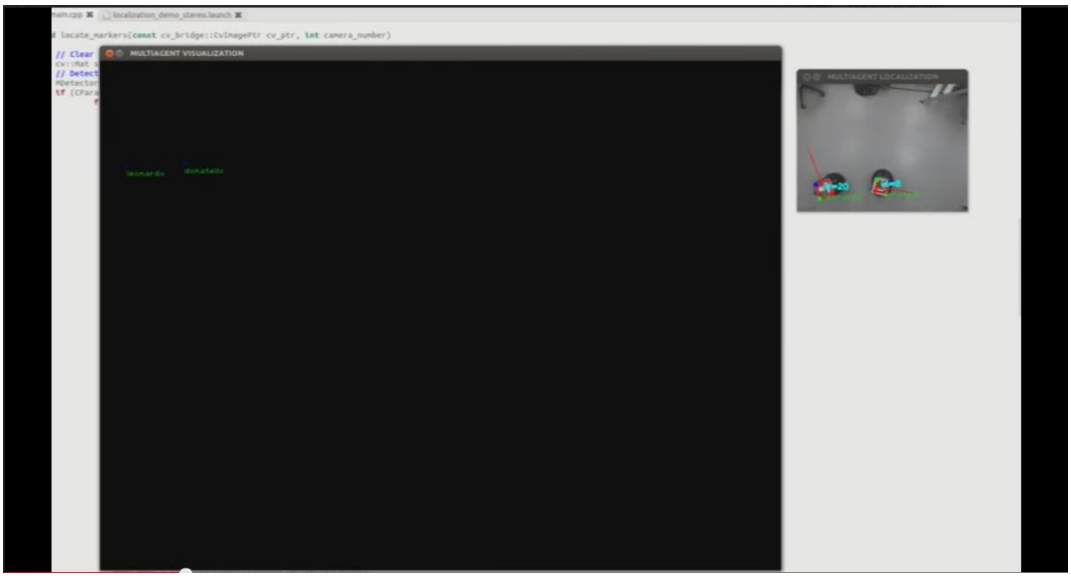

Figure 7: First camera

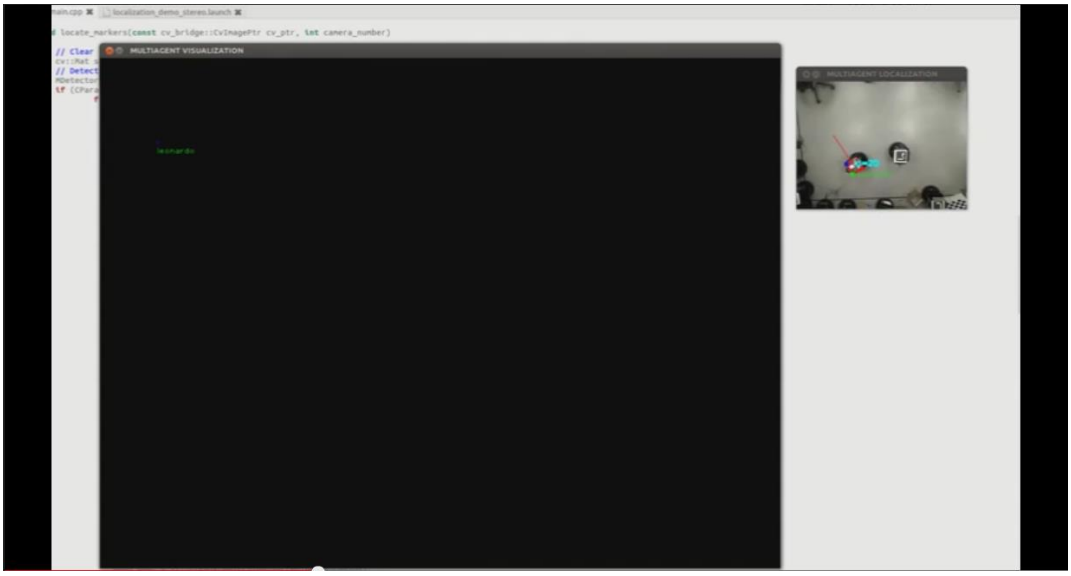

Figure 8: Second camera

4.3 XBOX Controller Integration

An XBOX 360 wireless controller was successfully integrated into the ROS system this quarter. To use, run `rosrun joy joy_node` on Calipso. Press the "X" symbol on the controller, and ensure that the wireless adapter is plugged in. To make the robot move around, use the left joystick on the XBOX controller. Make sure that you also depress the "LB" button on the top at the same time. For additional references, please refer to the GitHub wiki.

4.4 Interactive Demos

With the new developments in the lab this quarter, a few new opportunities for interactive demos have arisen.

4.4.1 Joystick Teleop

Goal: Demonstrate the coverage control algorithm

Implementation: Run the coverage control algorithm on as many robots as desired (2-3 is recommended). Run the joystick version of the control algorithm on the TurtleBot. A video of this demonstration can be found here.

4.4.2 Sensor Demonstration

Goal: Demonstrate the publisher/subscriber nature of ROS

Implementation: Give one participant the XBOX controller. Turn the first participant around so that they cannot see the TurtleBot. Let the second participant view the feed from the overhead camera, and the third participant view the view feed from the TurtleBot itself. The three should be encouraged to give verbal cues to each other to move to a goal position (as seen by the overhead camera), while avoiding obstacles (seen by both cameras). An example set up is shown in Figure X. The pink dotted line represents a possible path, while the red triangles are obstacles, which could be just trashcans.

On TurtleBot

```
roslaunch turtlebot_bringup minimal.launch
roslaunch turtlebot_bringup 3dsensor.launch
roslaunch turtlebot_deployment joystick_teleop_mstrcmptr.launch
```

On Calipso

```
roscore

rosrun joy joy_node

rosrun image_view image_view image:=/camera/rgb/image_color compressed
```

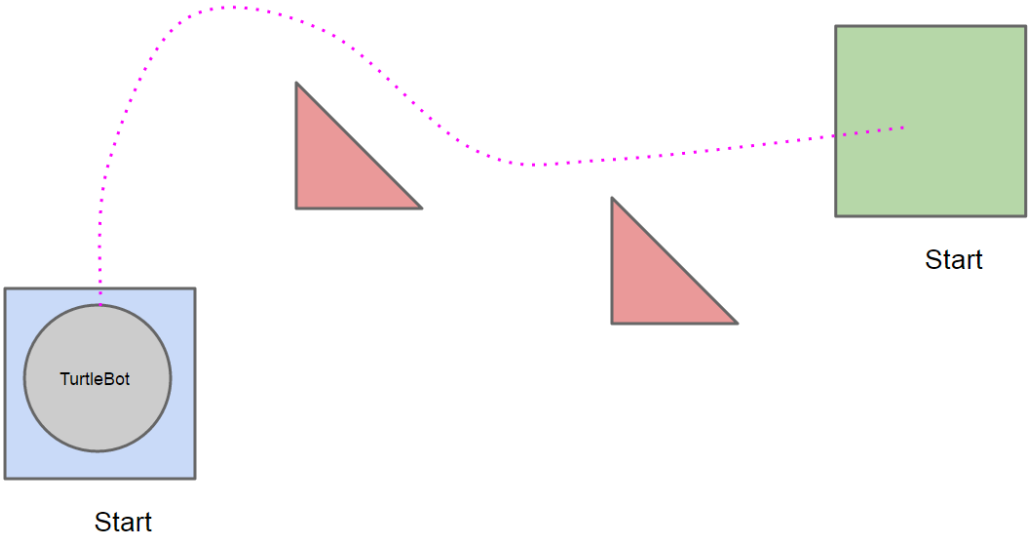

Figure X: Example of sensor demonstration set up

5 CONCLUSIONS

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

5.1 Troubleshooting/Tips

If the XBOX controller doesn't seem to be working, monitor the joystick topic to see if the message changes when a button is depressed. Also, ensure that the "LB" button is depressed *while* you are moving the joystick.

5.2 Path forward

In the next quarter, I plan on finishing the stereo vision node. The new camera needs to be installed, and then the system recalibrated. My goal is to complete the two camera system by end of the year, and begin testing.

6 REFERENCES

[1] Collins, Robert, and Penn State Cse486. Lecture 12: Camera Projection (n.d.): n. pag. Web. <http://www.cse.psu.edu/~rcollins/CSE486/lecture12.pdf>.

[2] "Camera Calibration and 3D Reconstruction." Camera Calibration and 3D Reconstruction — OpenCV 2.4.11.0 Documentation. OpenCV, n.d. Web. 20 Mar. 2015. <http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html>.

[3] "Essential Matrix." Wikipedia. Wikimedia Foundation, n.d. Web. 20 Mar. 2015. <http://en.wikipedia.org/wiki/Essential_matrix>.

[4] Collins, Robert, and Penn State Cse486. Lecture 13: Camera Projection II (n.d.): n. pag. Web. <http://www.cse.psu.edu/~rcollins/CSE486/lecture13.pdf>.