Robust Intelligent Sensing and Control Multi Agent Analysis Platform (RISC MAAP) Beginner Tutorial

D. Spencer Maughan[1]

I. Robotics Operating System (ROS)

ROS is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. Due to the many **features** and **integration** of powerful libraries it is not surprising that many of the world's top **engineers and researchers** make use of the many tools available through ROS. That is why here at Utah State University in the RISC Lab we are striving to become among the foremost contributers to this open source project to further interest and research in robotics.

II. Getting Started

- Install Ubuntu (**Installation Guide**, **Windows dual-boot**, **Mac dual-boot**)
  A CD or flash drive should be available upon request. If one is not available you may make your own. (**MAC**, **PC**)

- Install ROS (Select **Ubuntu** as your platform)

To get started in familiarizing yourself with ROS it is recommended beginning with the carefully constructed **tutorials** available online at ros.org. These will guide you through the many useful features available. Once you are familiar with the basics it will be much easier to understand the rest of this tutorial. However, I will strive to write this tutorial for the beginner that has little familiarity with Linux/Unix based systems or ROS.

III. Setting up a workspace

In the RISC Lab we all share the same workspace using apache subversion termed **SVN**. This enables us to make use of svn's many **features** to collaborate on projects, keep track of changes or even recurse back to old code if new designs were undesirable. To set up this shared workspace follow the following steps:

- source your ros environment.

  ```
  source /opt/ros/<devel>/setup.bash
  ```

  Fill in <devel> with whatever version of ros you installed. For example if you installed indigo:

  ```
  source /opt/ros/indigo/setup.bash
  ```

- Now follow these commands to create your workspace that can be found in ros tutorials

  ```
  mkdir -p ~/ros_workspace/src
  cd ~/ros_workspace/src
  catkin_init_workspace
  cd ~/ros_workspace/
  catkin_make
  cd ~
  nano .bashrc
  ```

  Now source your new workspace in the .bashrc file. This will ensure that your workspace is linked to ros every time a new terminal is opened.

  ```
  cd ~
  nano .bashrc
  ```

  add this line to the bottom.

  ```
  source ~/ros_workspace/devel/setup.bash
  ```

- checkout our repository (you will require a password from the gatekeeper (request who this is from Dr. Sharma)). Type these commands the terminal:

  ```
  cd ~/ros_workspace/src
  svn co svn://129.123.5.201/risc-lab
        /software/ros_workspace/current/src .
  ```

Make sure that the above IP address is accurate. It may change in the future. Before you can successfully make the packages in the newly formed workspace we must resolve some system dependencies

A. Resolving Dependencies

Run the following commands to resolve some dependency issues.

```
sudo apt-get install ros-indigo-joystick-drivers
daemontools libudev-dev libiw-dev
libblas-dev freeglut3-dev liblapack-dev
libopencv-dev libgsl0-dev python-scipy
libsdl1.2-dev openssh-server
rosdep install joy
sudo apt-get update
```

*This work was supported by the Department of Electrical Engineering at Utah State University

[1]D. Spencer Maughan is with the Department of Electrical and Computer Engineering, Utah State University, Logan, UT 84322, USA spence.maughan@gmail.com

```
rosdep update
```

Now you can catkin_make to build catkin packages and "rosmake -a" to build rosmake packages.

IV. Subversion

make a folder for your research:

update svn status commit with logs svn log -r 1:HEAD look at log of one individual svn log — sed -n '/USERNAME/,/—–$/ p' accept or reject version changes

V. File Navigation

I will not cover Navigation in this tutorial. Here are some resources for becoming familiar with navigating the **ROS** file system, **Terminal** environment and **Unix** in general.

VI. RISC Packages

Ros packages are merely folders that are linked to ros and can be used to **build** code into usable executable files. ROS Packages have a very specific **structure**. We have developed many packages in the RISC lab.

A. *cortex_ros:*

Allows communication with the **Motion Analysis Software** called cortex. It provides marker data and names of templates set up within that system.

B. *risc_control:*

Provides desired trajectories, frame of reference transformations, algorithms for robot controls.

C. *risc_visual:*

Provides RVIZ marker/3d model publishers for cortex visualization as well as image processing code using opencv in C++. i.e. Edge detection, optical flow, color thresholding, landmark detection.

D. *risc_simulator:*

Holds all of our simulations for systems of interest, fixed-wing aircraft, quadrotors, inverted pendulums.

E. *risc_estimation:*

Holds all of our estimation/filtering algorithms for estimating states such as kalman filtering, least squares or smoothing data using techniques such as as low pass filtering.

F. *risc_msgs:*

Contained within are descriptions for all of our custom messages. Some of these messages are as follows: 12 states, trajectory, bearing measurement, region of interest, markers, landmarks...

G. *risc_gui:*

This is intended to unify the many tools available in ROS that we intend to use in the lab. Ashish will provide details

H. *data_extraction:*

This package was originally authored by **Shane Lynn** and extracts data from a rosbag file and converts it to a csv file. I have modified it to include many of our custom messages. This is useful for providing data to those unfamiliar with ROS or for use in MATLAB. This would be unnecessary if we could fix the memory leak in our rosjava interface with matlab. I only hope someone reading this will take up the challenge. I have included a description of the ROS-MATLAB problems at the end of this tutorial.

VII. Other Packages:

The rest of these packages are all well documented on ROS or github websites.

A. **ardrone_autonomy***:*

Provides ardrone driver and navdata messages.

B. **ardrone_tutorials***:*

Provides some simple templates for interfacing with the ardrone and getting up and flying.

C. **roscopter***:*

Incorporates the **Mavlink** library for interfacing with the PixHawk (IRIS+)

D. **uvc_cam***:*

Enables you to publish your webcam video feed in ROS using the bf libuvc_camera driver. This is very useful for testing image processing code without needing an external camera.

E. **tum_ardrone***:*

Recent research package implementing SLAM and PTAM for autonomous navigation.

VIII. Using Motion Capture System

I set 200 Hz as the default camera rate.

A. *Calibration*

- Turn on cameras, Start cortex and Connect to cameras
- Setup Calibration
  - click on the"Calibrate..." button at the bottom left of the window
  - Select "Update Calibration" and click "Next"
  - click the "Check if volume is clean..." button and make sure all cameras are showing zero (you have to push play in the main window to visualize real time camera data)
    - If your volume is not clean, make sure to clear the space until it is clean.
    - If your volume is still not clean you need to mask or adjust camera settings (please ask for help)
  - If your volume is now clean. Prepare your wand with the 3 markers. Select "OK to Overwrite"
- Calibrate

  - When your wand-waver is ready click “Next” and wave that wand over the space of interest covering all axes of orientation(orient markers vertical some times and horizontal other times).
  - Keep waiving for about 2 minutes then click finish.
  - If it says the data is diverging the following may be problems:
    * you may not have a clean volume recheck to ensure that there are no extra visible markers/reflective surfaces/shoes/backpacks/or shiny metals.
    * your wand waver went crazy fast or didn’t cover the area, click previous and try again
    * the cameras have been over worked and may need some cool down time. Turn off the cameras and try again later.
    * the cameras may need some adjusting please ask for help
- The System is Calibrated.

B. Setting up a Template

Instructions:

- Determine a marker configuration that is asymmetrical but also provides you with a simple method of extracting all the desired information from your object such as: Position of the centroid, orientation...
- Create a recording of that object while it is stationary for about 10 seconds or so.
  - turn cameras on.
  - put the object of interest with the predetermined marker configuration in the middle of the MAAP.
  - launch Cortex and connect to the cameras.
  - Select Recording Settings... button.
  - Change Capture Duration to 10 seconds.
  - Name your recording then hit enter.
  - Click the bright red record button and wait until the recording is finished.
- Enter Post Processing environment and load your recording
- Create a new identifying template
  - click on the “New...” button next to MarkerSets.
  - name your identifying template and click enter.
- Name your markers use Quick ID.
  - First create a unique name for each marker that will be used in your template under the markers tab on the right.
  - click the “Quick ID” button and select the marker corresponding to each name.
- Link your named markers together
  - enter the “Links” tab and click the “Create Links for Template” button.
  - click and drag between points to create the links.
- Rectify (include all valid frames)
- Create Template
- Rectify Template
- Save
  - right click on the named tab and select save.

This really is a quick process and should take 5-10 minutes.

C. Streaming Raw Marker Data

To stream unfiltered raw marker positions with the associated template names run:

```
roscore
#  in separate terminal
rosrun cortex_ros stream_markers
```

This will be published under the topic name “/mocap_data”.

D. Streaming 12 State Estimates

To stream 12 state estimates of a marker configuration you first need to create a function to estimate your states in “estimation_functions.py” in the risc_estimation package. This function should follow the same naming as your template. If you want to use an estimation algorithm already set up then just add your template name in the list in the file “states_estimation.py” and ensure that your template will function given the setup.

Then run:

```
roscore
#  in separate terminal
rosrun risc_estimation stream states_estimation.p
```

This will be published under the topic name “/cortex” or “/landmarks” depending on which one you entered into your function. I have included an example under the section “Research Example”. I describe there how I calculate the twelve states given a marker setup.

IX. Visualizing Data

A. Rostopic

Rostopic is a tool in ros that provides a lot of relevant information about a specific topic.These features are:

```
rostopic bw     bandwidth used by topic
rostopic echo   print messages to screen
rostopic find   find topics by type
rostopic hz     publishing rate of topic
rostopic info   info about active topic
rostopic list   list active topics
rostopic pub    publish data to topic
rostopic type   print topic type
```

Running in terminal it is quick and simple. This is my preferred method while debugging or briefly inspecting performance.

B. rviz

rviz is a powerful 3d visualization tool in ROS. This can make your presentation videos look very professional and impressive. The code is written in C++. I have many examples available in risc_visual package. I will update these and make a launch file for them so that they are easy to use. I will discuss how to set up a demonstration launch file using this tool in the "Research Example" section.

C. rqt

rqt is a flexible GUI that integrates various plugins. Rqt makes it easier to manage all the various windows on the screen at one moment. If you become familiar with even a few of the many options available, you can visualize various message types in diverse graphs and plots to your hearts content.

D. Python

Python enables the generation of **plots** using **matplotlib**, **plotly** and many other **libraries**. This is also the language used to form many of the plugins available in rqt.

E. ardrone_gui

Ashish will likely fill in this space.

X. Python and C++

Here are some great **tutorials** for getting started in python. If you are already familiar with matlab you may benefit from looking over this useful **reference** outlining similarities between numpy and Matlab. Also, decent C++ tutorials can be found **here**. Python uses more of your cpu and is generally not as fast as C++. However, Python tends to be easier to use especially if you are already confortable in Matlab. RVIZ and cortex software applications are written in C++. Most everything else can use both C++ and Python.

XI. Publishing and Subscribing using Python/C++

I have provided some examples of publishers and subscribers using in the quad_command package or in C++ in risc_visual under the src folder. Look to those for examples. There are also **Python** and **C++** based tutorials on ros.org.

XII. Getting Data from Quadrotors

A. Ardrone

As soon as the ardrone_autonomy driver is running the navdata will be published as a rostopic. This data includes:

- batteryPercent: The remaining charge of the drone's battery (%)
- state: The Drone's current state: * 0: Unknown * 1: Inited * 2: Landed * 3,7: Flying * 4: Hovering * 5: Test (?) * 6: Taking off * 8: Landing * 9: Looping (?)
- rotX: Left/right tilt in degrees (rotation about the X axis)
- rotY: Forward/backward tilt in degrees (rotation about the Y axis)
- rotZ: Orientation in degrees (rotation about the Z axis)
- magX, magY, magZ: Magnetometer readings (AR-Drone 2.0 Only) (TBA: Convention)
- pressure: Pressure sensed by Drone's barometer (AR-Drone 2.0 Only) (Pa)
- temp : Temperature sensed by Drone's sensor (AR-Drone 2.0 Only) (TBA: Unit)
- wind_speed: Estimated wind speed (AR-Drone 2.0 Only) (TBA: Unit)
- wind_angle: Estimated wind angle (AR-Drone 2.0 Only) (TBA: Unit)
- wind_comp_angle: Estimated wind angle compensation (AR-Drone 2.0 Only) (TBA: Unit)
- altd: Estimated altitude (mm)
- motor1..4: Motor PWM values
- vx, vy, vz: Linear velocity (mm/s) [TBA: Convention]
- ax, ay, az: Linear acceleration (g) [TBA: Convention]
- tm: Timestamp of the data returned by the Drone returned as number of micro-seconds passed since Drone's boot-up

To publish this data ensure that the you are connected to the ardrone's wireless network then run:

```
roscore
rosrun ardrone_autonomy ardrone_driver
```

The preferred method would be to initialize this driver in a roslaunch file. This allows you to set the parameters of the ardrone.

B. IRIS+

In progress... Currently we can get data streaming around 3 Hz. We are working on increasing this.

XIII. Sending Data to Quadrotors

A. Ardrone

While the ardrone_driver is running the AR Drone is subscribing to the following topics:

- ardrone/takeoff
- ardrone/land
- ardrone/reset
  (Publishing an empty message to any of these three topics will result in the corresponding action.)

- cmd_vel
  This topic type is **geometry_twist**.
  The corresponding actions from inputs are as follows:

```
-linear.x: move backward
+linear.x: move forward
-linear.y: move right
+linear.y: move left
-linear.z: move down
+linear.z: move up

-angular.z: turn left
+angular.z: turn right
```

We have a class of functions set up in the quad_command package under risc_controller.py. This is currently how we are communicating to the AR Drone using the above method.

B. IRIS+

We are still working on it.

XIV. Recording and Saving Data

A. Using rosbag

rosbag is a set of tools for recording from and playing back to ROS topics. It is intended to be high performance and avoids deserialization and reserialization of the messages. It can be used in **command-line** as well as within **C++ or Python** code to store data. Current list of supported commands:

- **record** Record a bag file with the contents of specified topics.
- **info** Summarize the contents of a bag file.
- **play** Play back the contents of one or more bag files.
- **check** Determine whether a bag is playable in the current system, or if it can be migrated.
- **fix** Repair the messages in a bag file so that it can be played in the current system.
- **filter** Convert a bag file using Python expressions.
- **compress** Compress one or more bag files.
- **decompress** Decompress one or more bag files.
- **reindex** Reindex one or more broken bag files.

Rosbag recording is very fast and can handle large data storage at high rates if it is stored on the main hard drive. Otherwise, for example, if you are attempting to write a bag file through a USB port, you will likely notice lag for large data sets at high frequency (ex: 2+ HD video streams, sensor and cortex data).

B. Exporting data to csv file

edit file to manage whatever message you want...
rosrun data_extraction extract_all.py

XV. Simplifying Using Roslaunch

roslaunch is a tool for easily launching multiple ROS **nodes** locally and remotely via SSH, as well as setting parameters on the **Parameter Server**. It includes options to automatically respawn processes that have already died. roslaunch takes in one or more XML configuration files (with the .launch extension) that specify the parameters to set and nodes to launch, as well as the machines that they should be run on. Here is a **tutorial** that offers tips on creating a launch file that is reusable. I have also included an example demonstration launch file in the "Research Example" section of this tutorial.

XVI. real-time tuning

Explain trackbars using OpenCV

XVII. ROS Transforms

NOTE: if the transform listener rate is too high it will begin to lag behind the true data. Although the time stamp is accurate there is still a lag. Decreasing subscriber to 60 hertz seemed to remedy the problem NOTE: Lag still stacks with decreased rate. The fix that works best is to avoid while loops with a fixed rate rospy using rosy.spin() this will run the function as fast as possible every time it detects a published topic of interest. It is more smooth and stops appropriately

XVIII. ROS MATLAB INFO

NOTE: Do not print things in matlab from ros if your data rates are high. Matlab will print everything slowly and lag behind it is difficult to break the stream of data on the screen. ONLY print when debugging at low rates.

XIX. Camera Calibration

```
------Camera Calibration package------------------

rosdep update

rosdep install camera_calibration

rosmake camera_calibration

-------to run on ardrone-------
-------use the checkerboard in the closet
----Front camera
rosrun camera_calibration cameracalibrator.py --si
------bottom camera
rosrun camera_calibration cameracalibrator.py --si
```

XX. Ubuntu setup

```
--------------- Useful Software -----------------
sudo apt-get install terminator

gconftool --type string --set /desktop/gnome/appli

sudo apt-get install winetricks aptitude inkscape

winetricks vcrun2005

winetricks flash11

sudo apt-add-repository ppa:pipelight/stable

sudo apt-get update

sudo apt-get install pipelight-multi

pipelight-plugin --enable silverlight
```

```
sudo apt-get remove vim-tiny
sudo apt-get install vim

copy .config and .gconf over to the home directory from svn server
copy over .conkyrc
```

XXI. Productivity Hints

A. vim

Some useful Plugins and why

-

B. terminator

C. speedy vim-like navigation

D. Things to avoid in code

add list of cpp memory leaks and python speed drainers in ros cv::cvtColor function has a memory leak in it I have created a replacement function. NOTE: Do not subscribe to multiple topics if they are being published at significantly different rates! For example: Cortex data comes at around 200 Hz and Camera data from the quad comes at 30 Hz if they are subscribed to in the same script everything will be published at ¡30Hz

I spent too much time hunting for a ”ready-made” function when I could have coded it myself much faster.

XXII. Research Example

provide step by step implementation of flying waypoints or maybe flying a quad with a simulated pendulum on top.

A. Template

get picture of quad label markers 1-3. get picture of template on cortex label markers 1-3

B. State Estimation

show python code for estimation “estimation_functions.py” as well as “states_estimation.py”

C. Subscribing/Publishing

show trajectory code snippet as well as controller code snippet explain/show risc_controller code snippet

make diagram for joystick control

D. Setting up Launch File Demonstration

show launch file snippet briefly summarize rviz marker code and how it functions in launch file

show screen shot of rviz demo