Robot Operation System

Lecture material based on the book, ROS Robot Programming

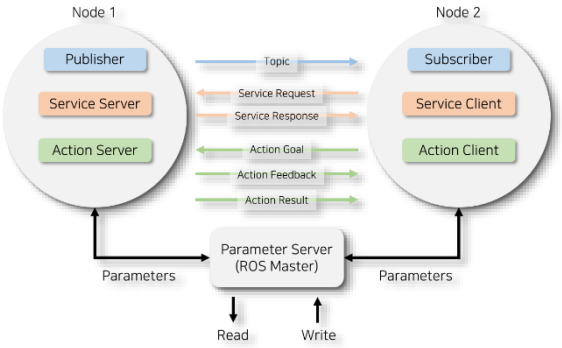
2018 Fall

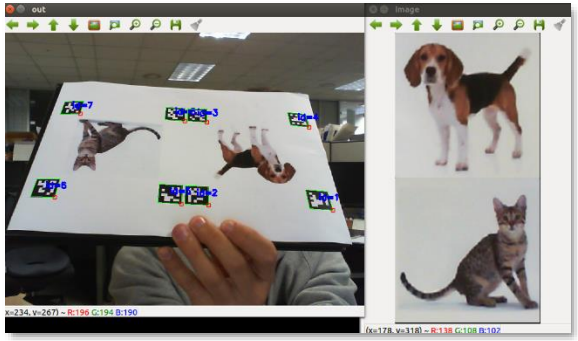
ME401 Capstone Design
ME491 Programming for Autonomous Mobile System

KAIST Department of Mechanical Engineering

T.A. Dongha Chung chungdongha@kaist.ac.kr

CONTENTS





Some theoretical background + Code explanation

Do not just copy & paste the code without understanding them

`Is for terminal command`

`Is for terminal output`

`Is for script`

Prerequisite

New and shiny is not always the best

ubuntu®

16.04.5 Xenial Xerus – [Eng]

+

ROS Kinetic

How to setup ubuntu 16.04.5 OS :
1. Download iso file from http://releases.ubuntu.com/16.04/
2. Make bootable usb : https://tutorials.ubuntu.com/tutorial/tutorial-create-a-usb-stick-on-windows#3
3. Setup : https://tutorials.ubuntu.com/tutorial/tutorial-install-ubuntu-desktop#4

How to setup ROS in ubuntu :
1. Follow instruction and select "**Desktop-full install**" : http://wiki.ros.org/kinetic/Installation/Ubuntu

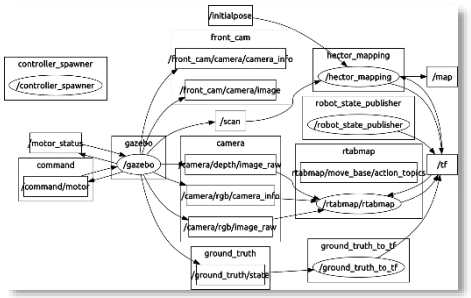
※It is important to install ROS with "Desktop-full install" option

---

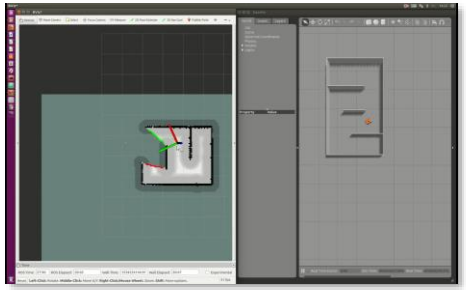
Additional (recommended for your convenience) setup
Ubuntu hangul(한글) : http://hochulshin.com/ubuntu-1604-hangul/
Atom (a code editing program) : https://codeforgeek.com/2014/09/install-atom-editor-ubuntu-14-04/
Terminator (splitable termianl) : https://askubuntu.com/questions/829045/how-do-i-install-terminator

Introduction to ROS

ROS is an open-source, meta-operating system for your robot



Plumbing

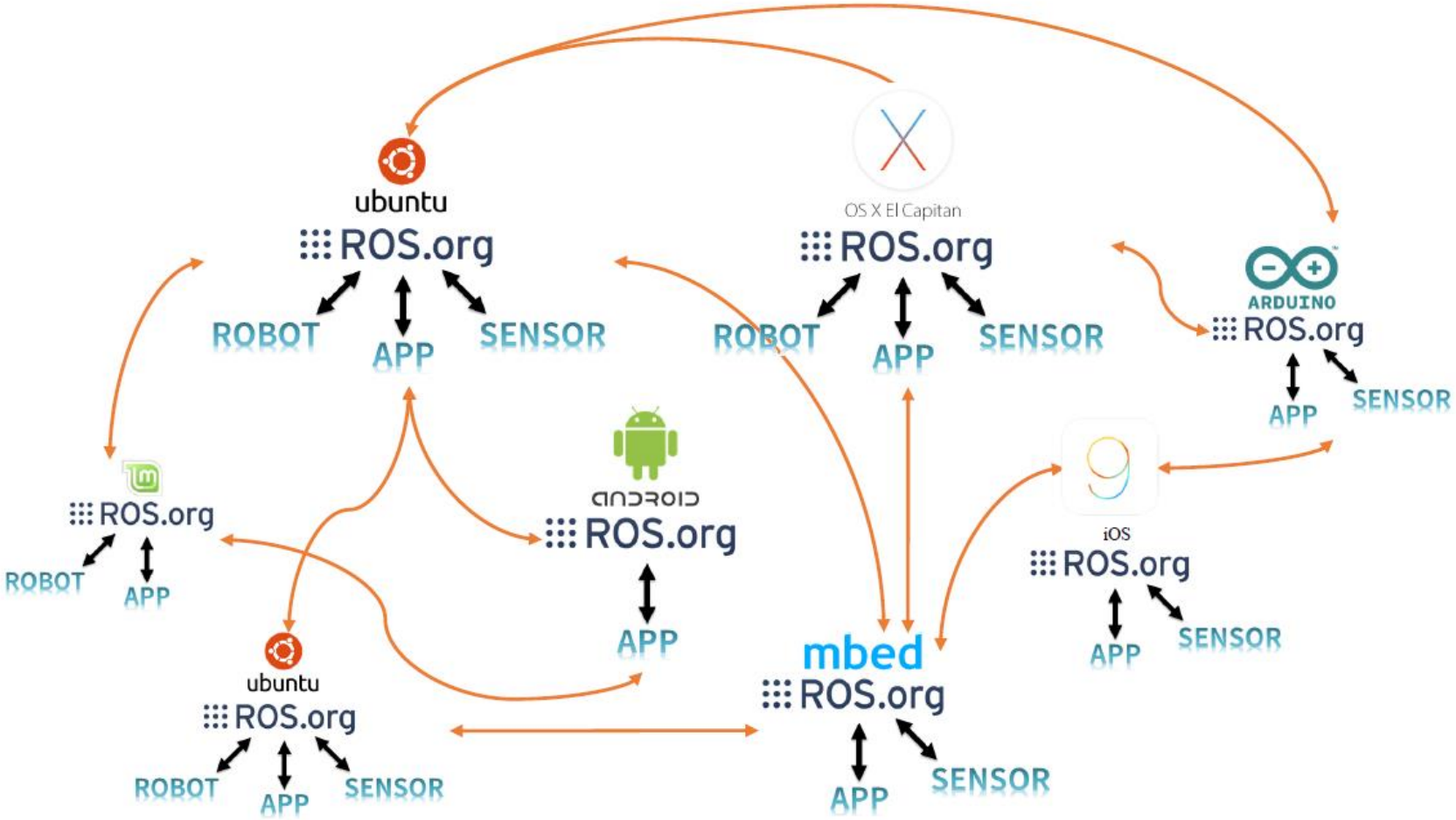


Tools

Algorithms

Introduction to ROS

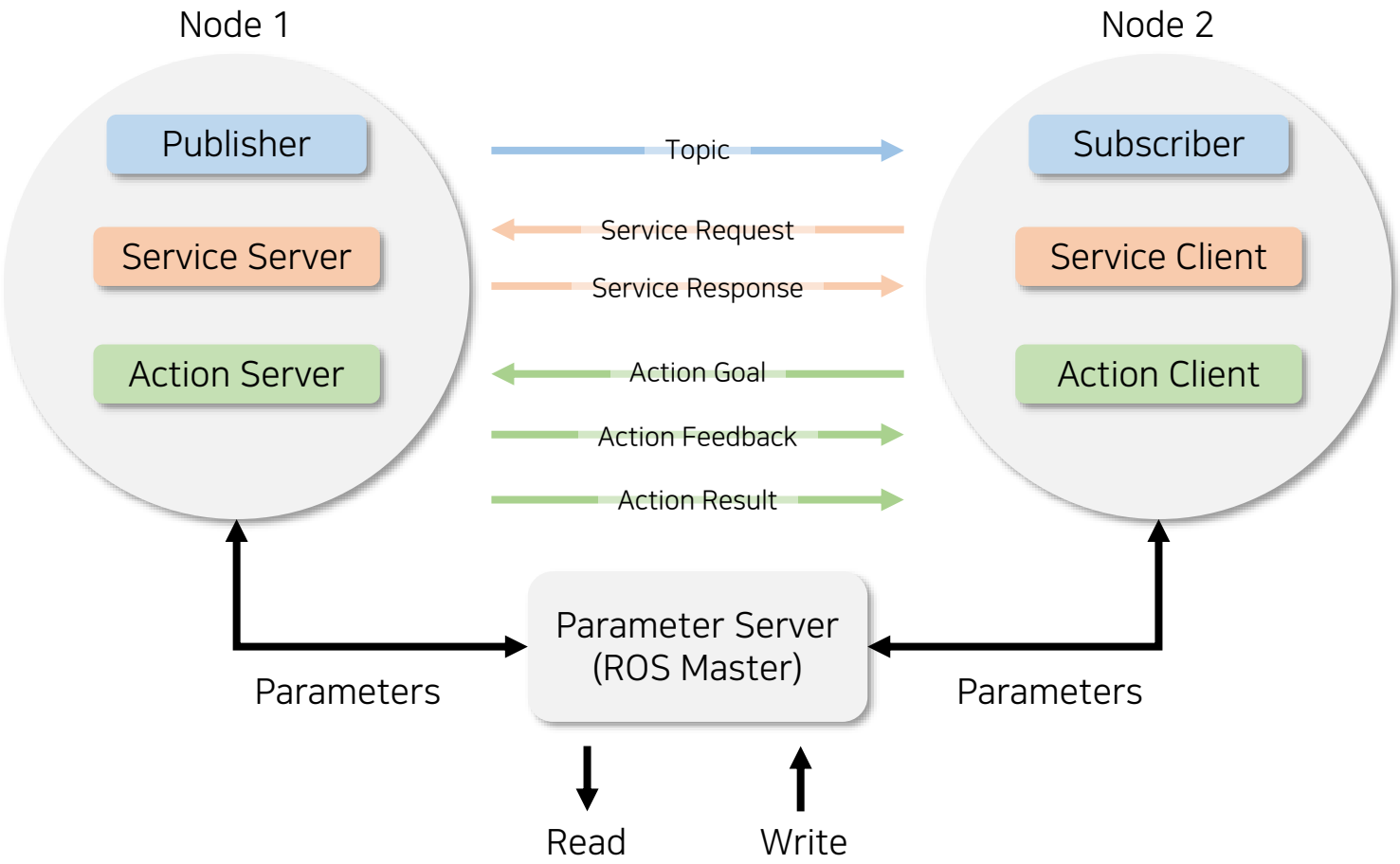
Compatible with different OS



Making nodes and connecting them

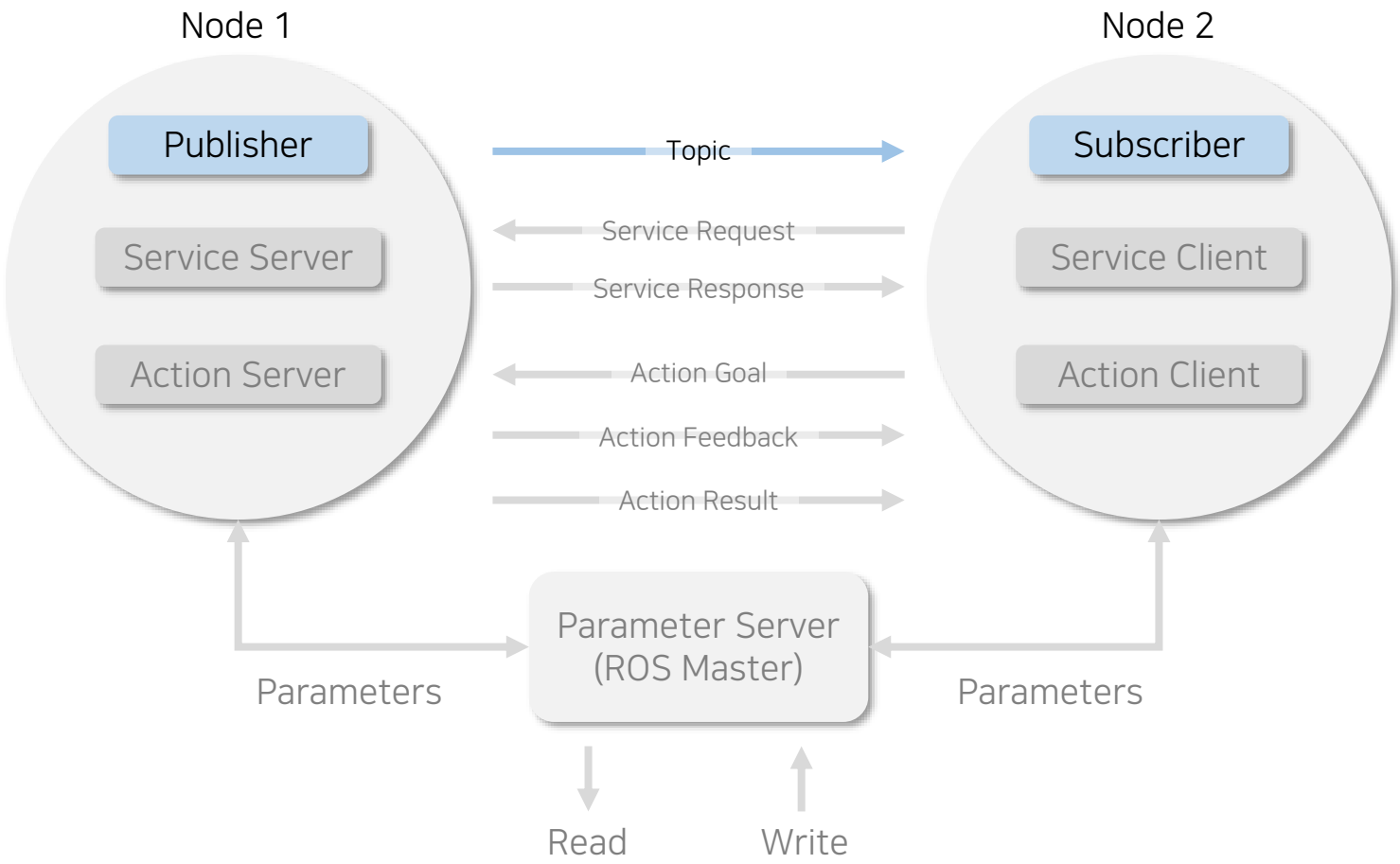
Communications between nodes & ROS master

Chapter 7 : https://github.com/robotpilot/ros-seminar/blob/master/07_ROS_기본_프로그래밍.pdf



Making nodes and connecting them

Publish & Subscribe between nodes



Making nodes and connecting them

Let's start with making workspace

```
~$ cd                                                    -  Goes to home directory
~$ mkdir catkin_make                                     -  Make a folder named "catkin_ws"
~$ cd catkin_make                                        -  Goes inside the folder "capstone_simulation"
~$ mkdir src                                             -  Important! : the folder name should be "src"
~$ catkin_make init                                      -  Initialize the workspace
~$ echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc  -  Include the bash file in workspace
~$ source ~/.bashrc                                      -  Source ~/.bashrc
```

Now, let's create a package named "ros_tutorials_topic"

You are now at ~/catkin_ws

```
~$ cd src                                                -  Goes to src
~$ catkin_create_pkg ros_tutorials_topic message_generation std_msgs roscpp
```

Package name

Dependency 1 "We are going to generate messages"

Dependency 2 "We are going to use standard messages"

Dependency 3 "We are going to use c++ language"

Making nodes and connecting them

Let's start with making workspace

You are now at ~/catkin_ws/src

```
~$ cd ros_tutorials_topic      -  Goes to ros_tutorials_topic
~$ ls                          -  List segments
```

You will see :

```
include                        -  Folder containing header files
src                            -  Folder containing source codes
CMakeLists.txt                 -  File for build settings
package.xml                    -  File for package settings
```

Make some changes in package.xml (Optional)

```
~$ gedit package.xml           -  Open file using text modifying program 'gedit'
```

Making nodes and connecting them

package.xml

Description of the package

```
<?xml version="1.0"?>
<package format="2">
<name>ros_tutorials_topic</name>
<version>0.0.0</version>
<description>The ros_tutorials_topic package</description>
<maintainer email="chungdongha@kaist.ac.kr">Dongha Chung</maintainer>
<license>TODO</license>  <buildtool_depend>catkin</buildtool_depend>
<build_depend>message_generation</build_depend>
<build_depend>roscpp</build_depend>
<build_depend>std_msgs</build_depend>
<build_export_depend>roscpp</build_export_depend>
<build_export_depend>std_msgs</build_export_depend>
<exec_depend>roscpp</exec_depend>
<exec_depend>std_msgs</exec_depend>
<export>  </export>
</package>
```

Save & exit

Make some changes in CMakeLists.txt (Not optional)

```
~$ gedit CMakeLists.txt                    -  Open file using text modifying program ‘gedit’
```

Making nodes and connecting them

CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8.3)
project(ros_tutorials_topic)

## Packages required for catkin build
## If the dependent packages (message generation roscpp std_msgs) does not exist : error
find_package(catkin REQUIRED COMPONENTS message_generation roscpp std_msgs)
## Declare message : MsgTutorial.msg
add_message_files(FILES MsgTutorial.msg)
## Set the dependent message : std_msgs
generate_messages(DEPENDENCIES std_msgs)
## Options for catkin package : dependencies on the system and catkin build
catkin_package(
  LIBRARIES ros_tutorials_topic
  CATKIN_DEPENDS std_msgs roscpp)
## Setting the include directories
include_directories(${catkin_INCLUDE_DIRS})
## Options for the node, 'topic_publisher'
## Settings for executable file, target link library, and additional libraries
add_executable(topic_publisher src/topic_publisher.cpp)
add_dependencies(topic_publisher ${${PROJECT_NAME}_EXPORTED_TARGETS}${catkin_EXPORTED_TARGETS})
target_link_libraries(topic_publisher ${catkin_LIBRARIES})
## Options for the node, 'topic_subscriber'
add_executable(topic_subscriber src/topic_subscriber.cpp)
add_dependencies(topic_subscriber ${${PROJECT_NAME}_EXPORTED_TARGETS}${catkin_EXPORTED_TARGETS})
target_link_libraries(topic_subscriber ${catkin_LIBRARIES})
```

[1]

[2]

[3]

Save & exit

Making nodes and connecting them

Making messages : [1]

```
## Declare message : MsgTutorial.msg
add_message_files(FILES MsgTutorial.msg)
```

[1]

```
                                        You are now at ~/catkin_ws/src/ros_tutorials_topic
~$ mkdir msg                              -  Make a directory named msg
~$ cd msg
~$ gedit MsgTutorial.msg                  -  Create a msg file named MsgTutorial
```

MsgTutorial.msg

```
time stamp                                -  time message named ‘stamp’
int32 data                                -  int32 message named ‘data’
```

```
Save & exit
```

Making nodes and connecting them

Making publisher node [2]

```
## Options for the node, 'topic_publisher'
add_executable(topic_publisher src/topic_publisher.cpp)
```

[2]

```
~$ cd ..                                          -  Goes back to upper directory
~$ cd src
~$ gedit topic_publisher.cpp                      -  Create a cpp file named topic_publisher
```

topic_publisher.cpp

```
#include "ros/ros.h"                              -  Basic ROS header file
#include "ros_tutorials_topic/MsgTutorial.h"      -  MsgTutorial header file
int main(int argc, char **argv)                   -  Main function of the node
{
  ros::init(argc, argv, "topic_publisher");       -  Initialize the node name as "topic publisher"
  ros::NodeHandle nh;                             -  Declare the node handle for communication
                                                     with ROS system

  ros::Publisher ros_tutorial_pub = nh.advertise<ros_tutorials_topic::MsgTutorial>("ros_tutorial_msg", 100);
                                                  -  Declare the publisher 'ros_tutorial_pub :
                                                       -  We will use MsgTutorial message
                                                       -  Publisher queue size is 100

  ros::Rate loop_rate(10);                        -  Set the loop rate as 10 : 10Hz
  ros_tutorials_topic::MsgTutorial msg;           -  Declare a message variable msg
  int count = 0;                                  -  Initialize the variable 'count'

  while(ros::ok())                                -  While the node is running do :
  {…}
…
```

Making nodes and connecting them

Making publisher node [2]

topic_publisher.cpp

```
...
  while(ros::ok())
  {
    msg.stamp = ros::Time::now();                  - Set the message ‘stamp’ as current time
    msg.data = count;                              - Set the message ‘data’ as ‘count’
    ROS_INFO("send msg = %d", msg.stamp.sec);      - Print out stamp.sec
    ROS_INFO("send msg = %d", msg.stamp.nsec);     - Print out stamp.nsec
    ROS_INFO("send msg = %d", msg.data);           - Print out data
    ros_tutorial_pub.publish(msg);                 - Publish the message
    loop_rate.sleep();                             - Goes into sleep regarding to the loop rate
    ++count;                                       - Add 1 to variable ‘count’
  }
  return 0;
}
```

Save & exit

Making nodes and connecting them

Making subscriber node [3]

```
## Options for the node, 'topic_subscriber'
add_executable(topic_subscriber src/topic_subscriber.cpp)
```

[3]

topic_subscriber.cpp

```
#include "ros/ros.h"
#include "ros_tutorials_topic/MsgTutorial.h"

void msgCallback(const ros_tutorials_topic::MsgTutorial::ConstPtr& msg)
{                                                    -  Callback function which will run when
  ROS_INFO("recievemsg= %d", msg->stamp.sec);           message is received
  ROS_INFO("recievemsg= %d", msg->stamp.nsec);
  ROS_INFO("recievemsg= %d", msg->data);
}

int main(int argc, char **argv)                      -  Main function of the node
{
  ros::init(argc, argv, "topic_subscriber");         -  Initialize the node name as 'topic_subscriber'
  ros::NodeHandle nh;                                -  Declare node handle
  ros::Subscriber ros_tutorial_sub= nh.subscribe("ros_tutorial_msg", 100, msgCallback);
  ros::spin();                                       -  Declare the subscriber 'ros_tutorial_sub :
  return 0;                                             -  We will use MsgTutorial message
}                                                       -  Subscriber queue size is 100
```

Save & exit

Making nodes and connecting them

Build the package & run

```
~$ cd ~/catkin_ws                                   -  Goes to catkin_ws directory
~$ catkin_make                                      -  Execute catkin build
```

On one terminal

```
~$ roscore                              -  Run ROS Master, parameter server, and rosout logging node
```

On other terminal

```
~$ rosrun ros_tutorials_topic topic_publisher    -  Run the ros node topic_publisher
```

Package name — Node name

On another terminal

```
~$ rosrun ros_tutorials_topic topic_subscriber    -  Run the ros node topic_subscriber
```

Package name — Node name

On another terminal

```
~$ rqt_graph                                        -  Shows the nodes and their connections
```

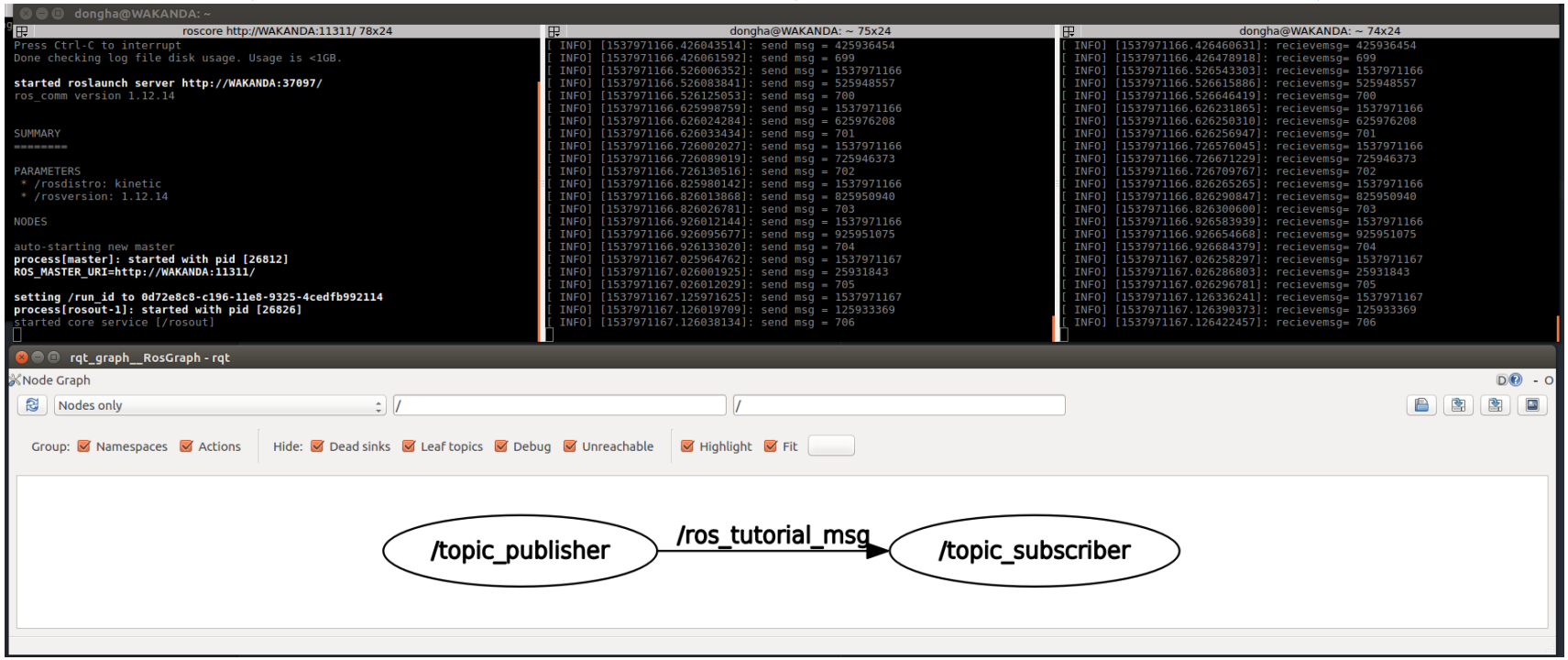
Making nodes and connecting them

Output

roscore

topic_publisher node

topic_subscriber node



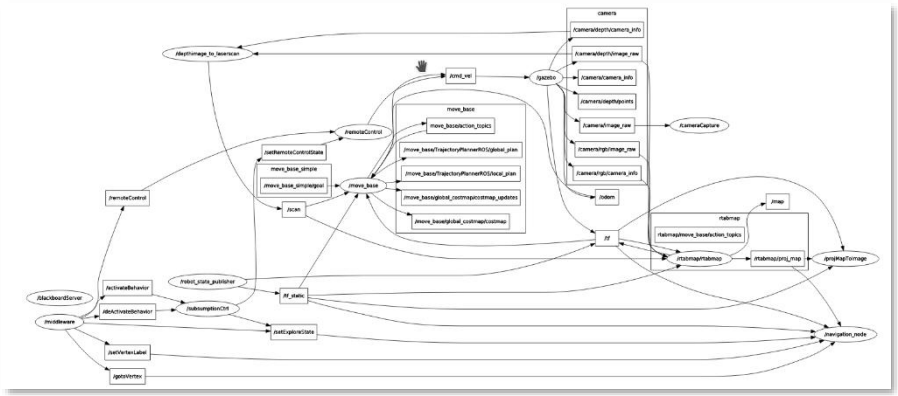
rqt_graph

Making nodes and connecting them

Publish & Subscribe between nodes

For much complex systems…

Node1, Node2, Node3, … Node n [1]



Publish-Subscribe relationship (rqt_graph) [2]

[1] https://videohive.net/item/black-lines-and-dots-clean-background/16910801
[2] https://medium.com/@genefoxwell/marvins-head-pt-1-faf260831883

For more information…

Dr. Pyo's lecture note

https://github.com/robotpilot/ros-seminar

Getting the cat image and dog image from webcam

Problem with using CNN for webcam image

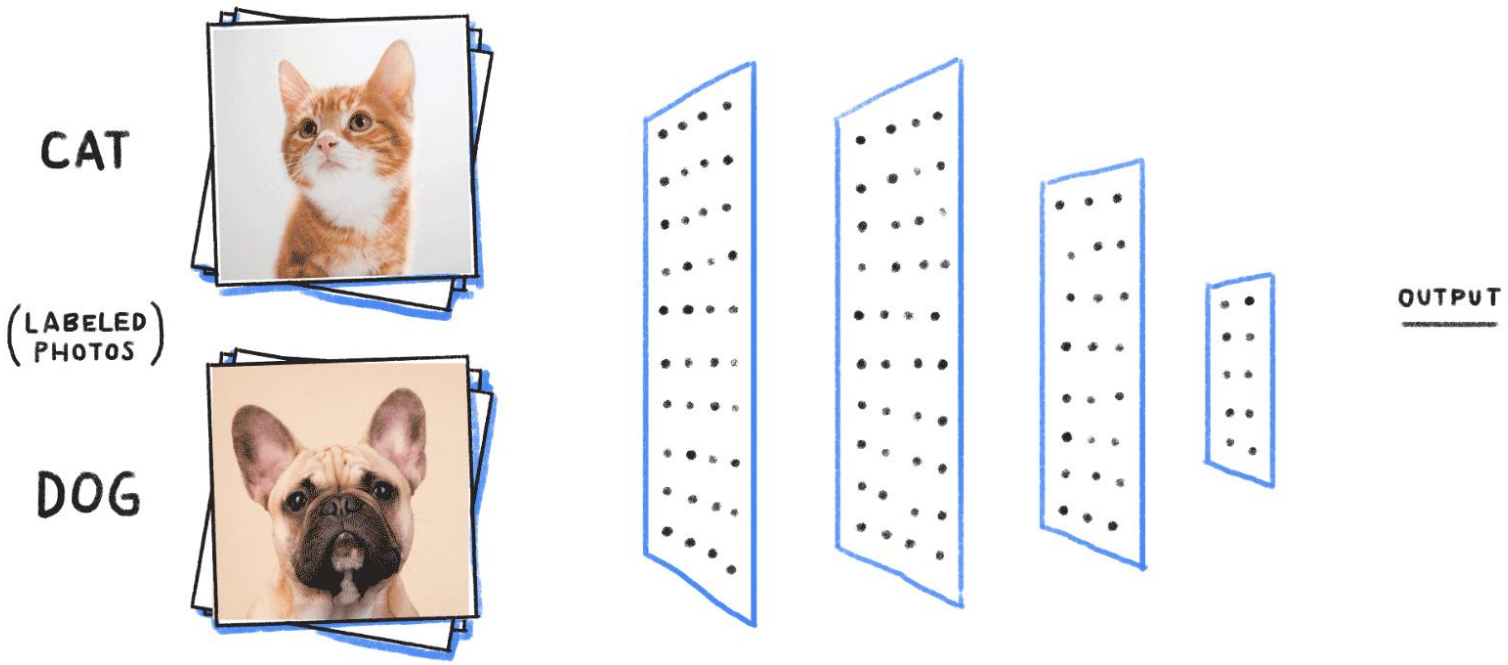
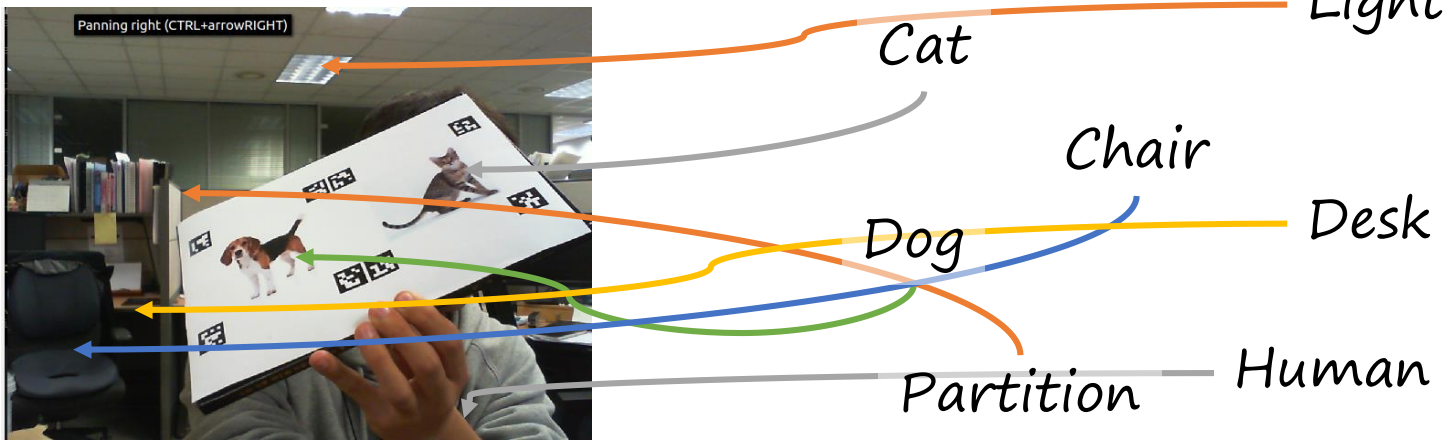


Image from https://becominghuman.ai/building-an-image-classifier-using-deep-learning-in-python-totally-from-a-beginners-perspective-be8dbaf22dd8

Getting the cat image and dog image from webcam

Problem with using CNN for webcam image



Too many information in one image

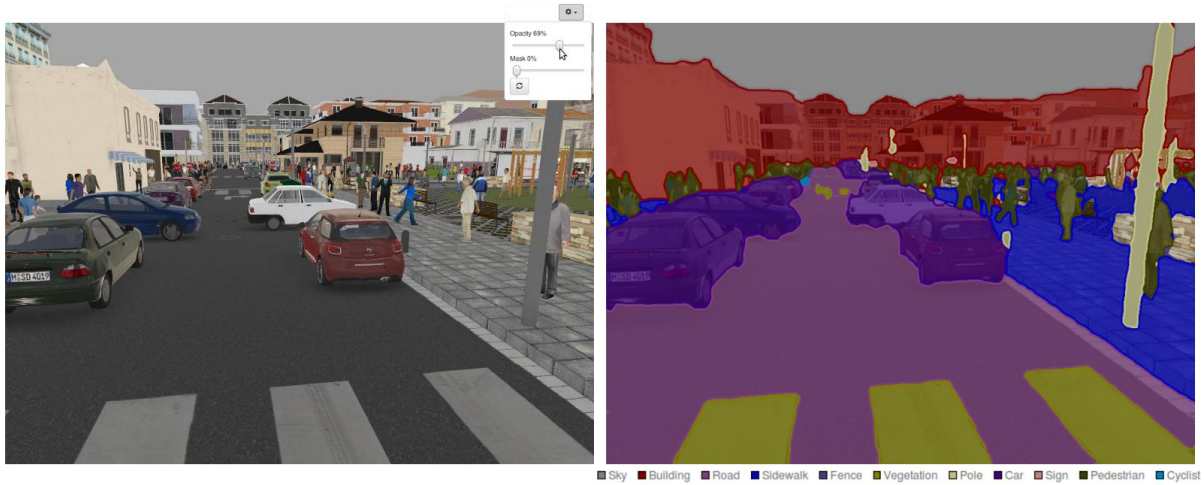


Image segmentation : https://medium.com/nanonets/how-to-do-image-segmentation-using-deep-learning-c673cc5862ef

Assignment : Getting warped cat & dog images from webcam

Getting the cat image and dog image from webcam

From webcam to readmarker

Capture a video frame from webcam 0 (if it does not work, try 1, 2, 3…)

```
VideoCapture cap = cv::VideoCapture(0);
```

Convert the image into Mat format

```
cap >> frame;
```

Package “webcam”

Node “image_publisher”

Package “readmarker”

Node “marker_reader”

Getting the cat image and dog image from webcam

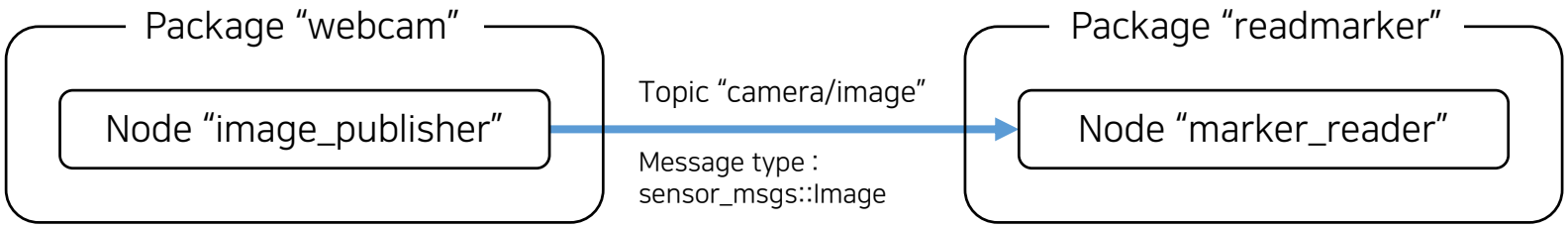
From webcam to readmarker

Convert the OpenCV Mat format to ROS message

```
msg = cv_bridge::CvImage(std_msgs::Header(), "bgr8", frame).toImageMsg();
```

Publish the message

```
pub.publish(msg);
```



Getting the cat image and dog image from webcam

From webcam to readmarker

Crop and warp the image when the image is subscribed

```
void imageCallback(const sensor_msgs::ImageConstPtr& msg);
```

Convert the incoming ROS image message to OpenCV CvImage

```
cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::image_encodings::BGR8);
```

Convert the OpenCV CvImage to Mat

```
cv_ptr->image.copyTo(inputImage);
```

Detect markers, crop, warp, & publish

```
...
```

Package “webcam”

Node “image_publisher”

Package “readmarker”

Node “marker_reader”

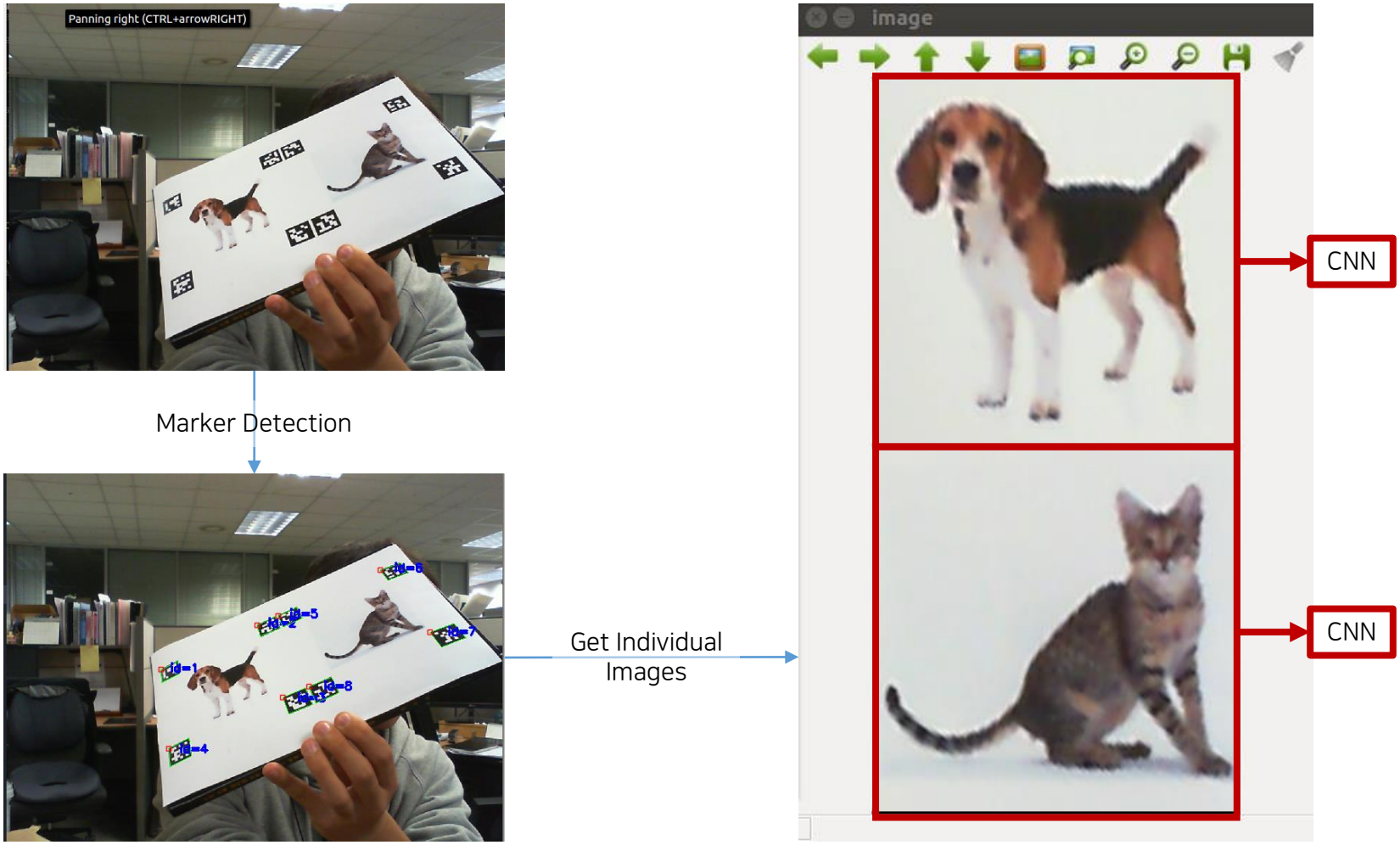
Topic “cropped_img”

Message type :
1. int32
2. sensor_msgs::CompressedImage

Out

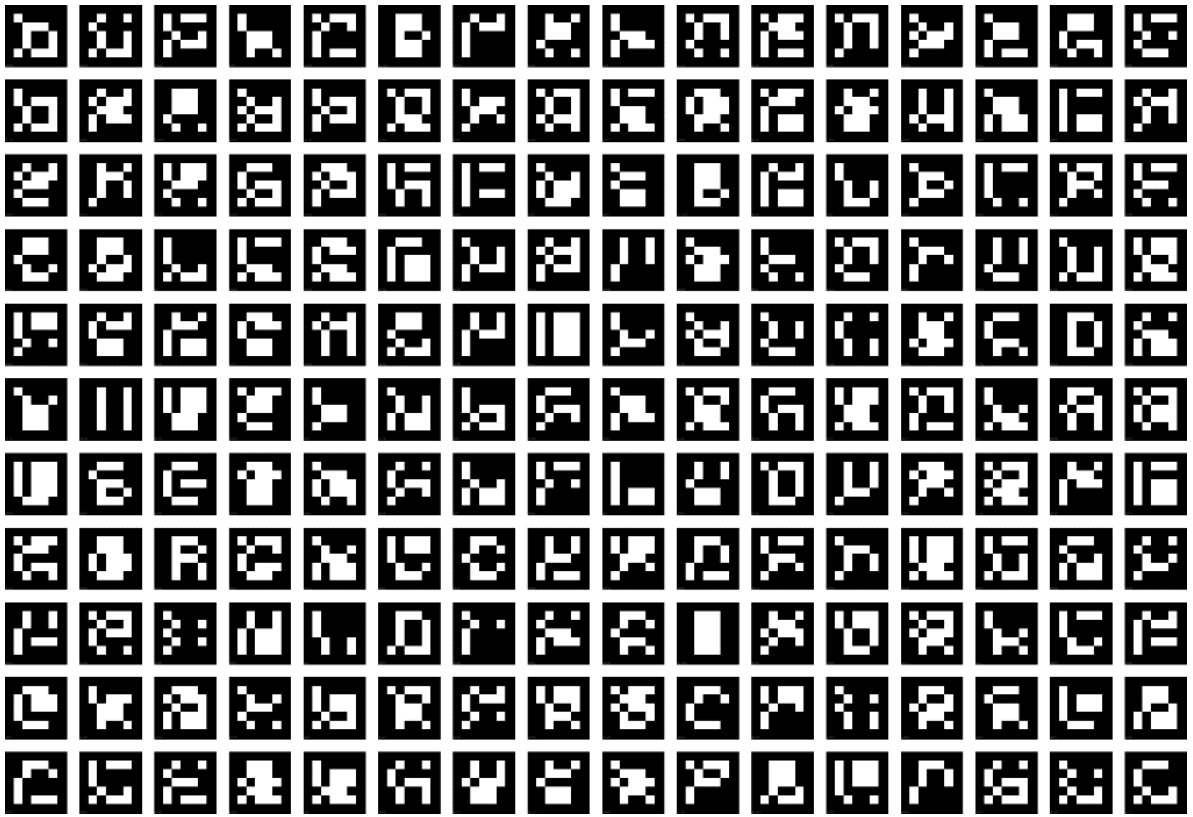
Getting the cat image and dog image from webcam

Use the AruCo markers to crop the images



Getting the cat image and dog image from webcam

AruCo in OpenCV



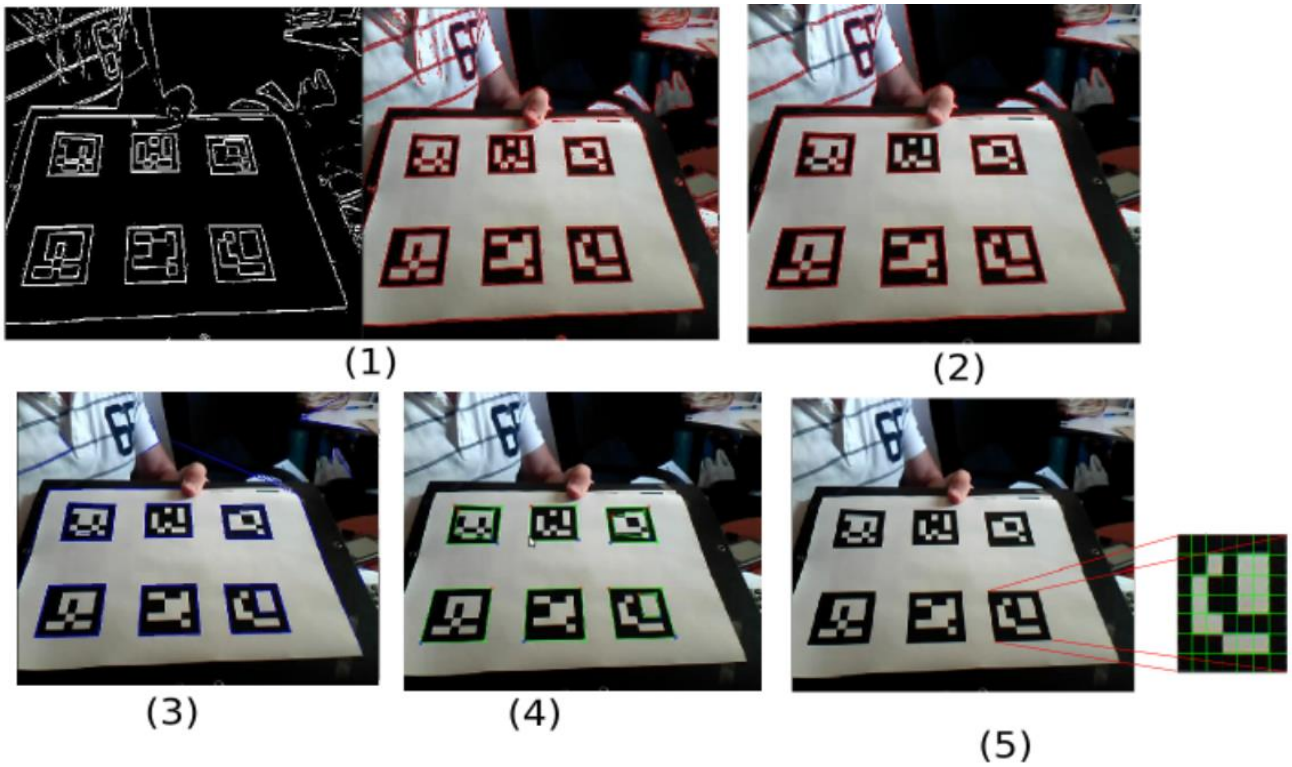
AruCo markers

Predefined marker dictionary

Making AruCo markers -> see package aruco

Getting the cat image and dog image from webcam

AruCo in OpenCV



1. Apply an Adaptive Thresholding so as to obtain borders
   Find contours. After that, not only the real markers are detected but also a lot of undesired borders.
2. Remove borders with an small number of points
3. Polygonal approximation of contour and keep the concave contours with exactly 4 corners
   Sort corners in anti-clockwise direction
4. Remove too close rectangles
5. Marker Identification

[1] https://docs.google.com/document/d/1QU9KoBtjSM2kF6lTOjQ76xqL7H0TEtXriJX5kwi9Kgc/edit#

Getting the cat image and dog image from webcam

Use the AruCo markers to crop the images

Marker Detection

Detect the markers

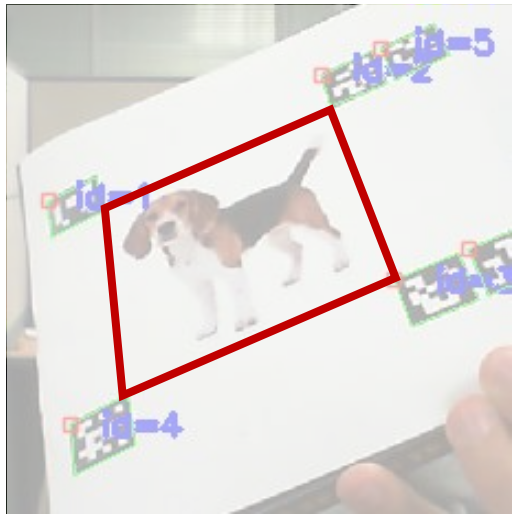
```
aruco::detectMarkers(inputImage, dictionary, markerCorners, markerIds)
```

Check if markers 1~4 & 5~8 are detected

```
b_image1 and b_image2 is used to check if the images are available
```

If image 1 (marker 1~4) is available : b_image1 = 1,
Get the coordinates of image 1 coordinates and find homography

```
if(b_image1==1){…}
```



Getting the cat image and dog image from webcam

Use the AruCo markers to crop the images

Marker Detection

Detect the markers

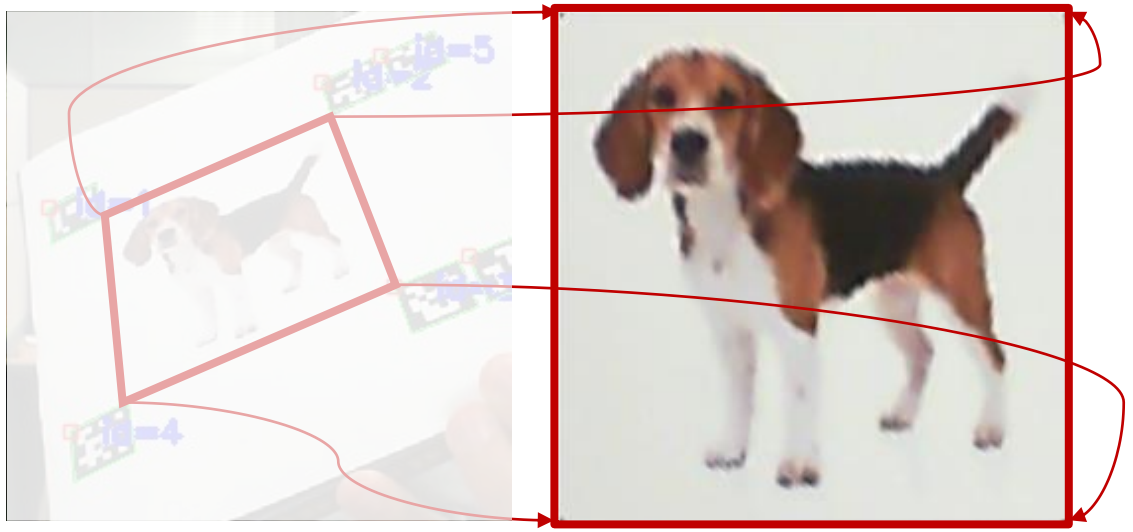
```
aruco::detectMarkers(inputImage, dictionary, markerCorners, markerIds)
```

Check if markers 1~4 & 5~8 are detected

```
b_image1 and b_image2 is used to check if the images are available
```

If image 1 (marker 1~4) is available : b_image1 = 1,
Get the coordinates of image 1 coordinates and find homography

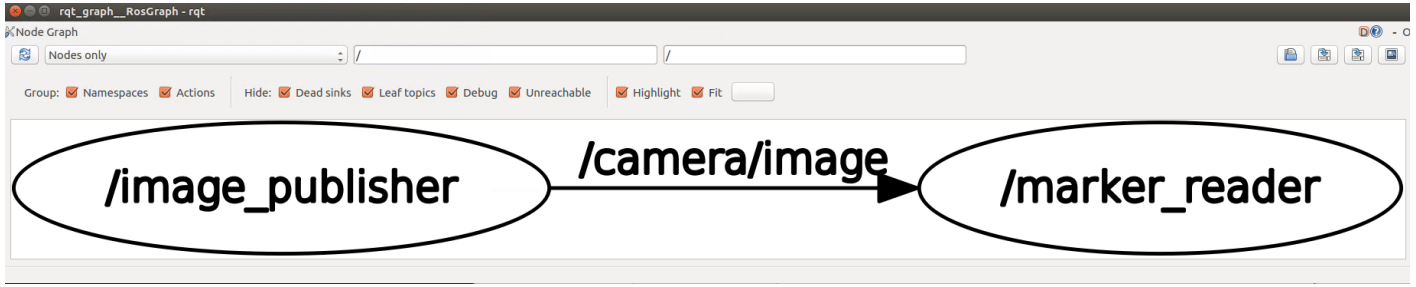
```
if(b_image1==1){…}
```



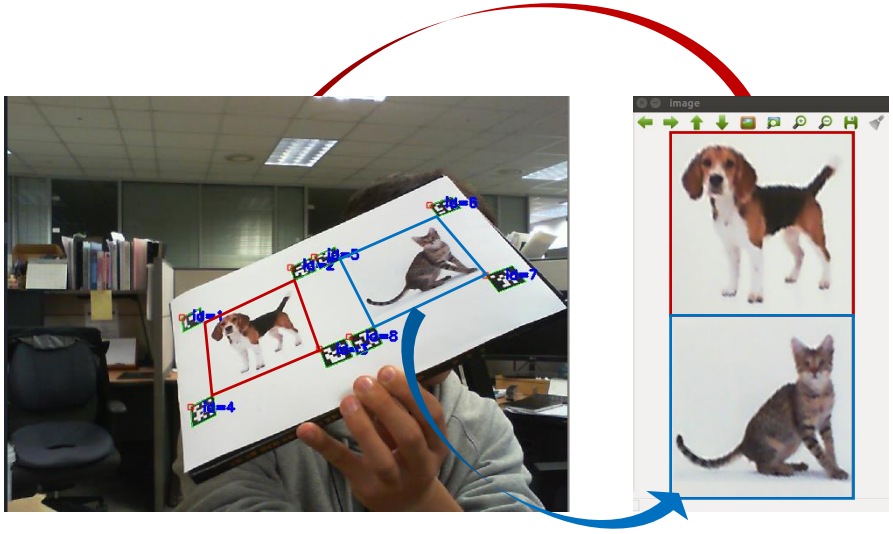
Homography (transformation matrix) : $\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = H \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$

In this assignment

1. Know how the images are published / subscribed



2. Know how the dog image and cat image is cropped and warped



Review codes : webcam.cpp & readmarker.cpp

Appendix : Sample image

1

2

5

6

4

3

8

7