

# ROS Robot Operation System

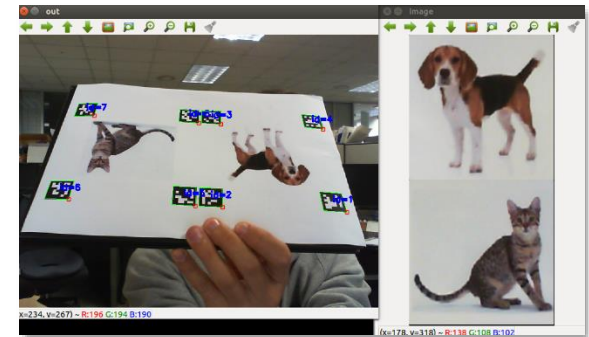
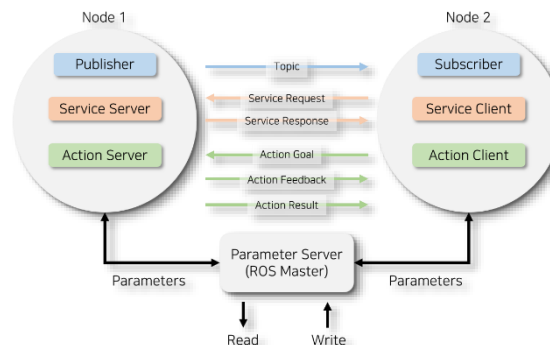
Lecture material based on the book, [ROS Robot Programming](#)

2018 Fall

ME401 Capstone Design  
ME491 Programming for Autonomous Mobile System

# CONTENTS

- 01. Prerequisite : Installing Ubuntu 16.04.5 & ROS kinetic
- 02. Extremely brief introduction to ROS
- 03. Making nodes, publisher & subscriber
- 04. Assignment : Getting warped cat & dog images from webcam



Some theoretical background + Code explanation

Do not just copy & paste the code without understanding them

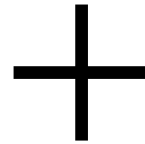
Is for terminal command

Is for terminal output

Is for script

# Prerequisite

New and shiny is not always the best



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

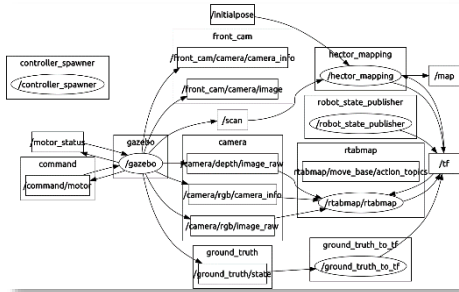
Ubuntu hangul(한글) : <http://hochulshin.com/ubuntu-1604-hangul/>

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

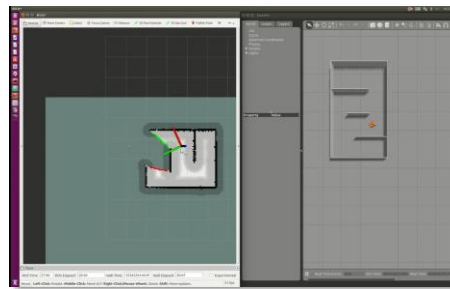
ROS is an open-source, meta-operating system for your robot



Plumbing



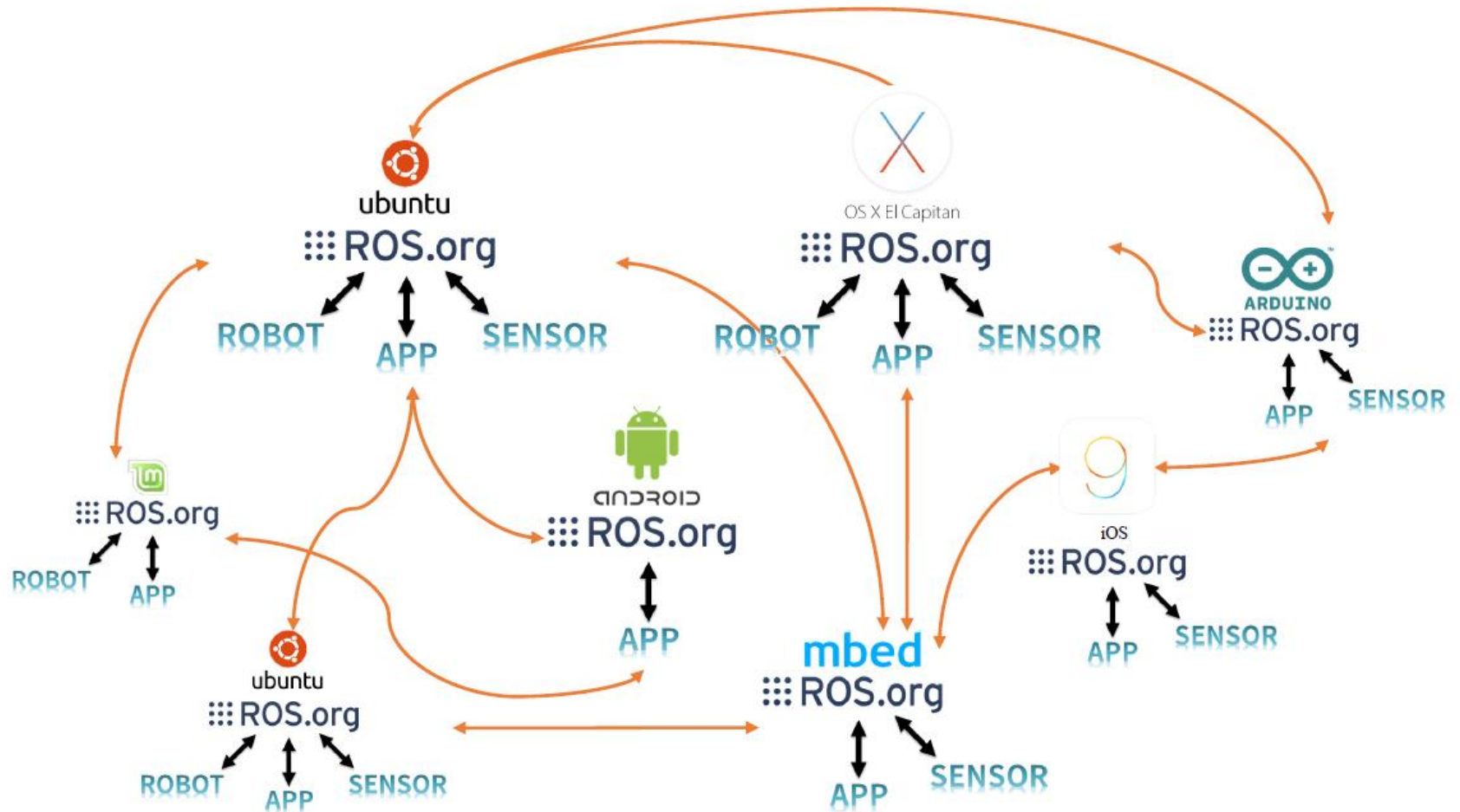
Algorithms



Tools

# Introduction to ROS

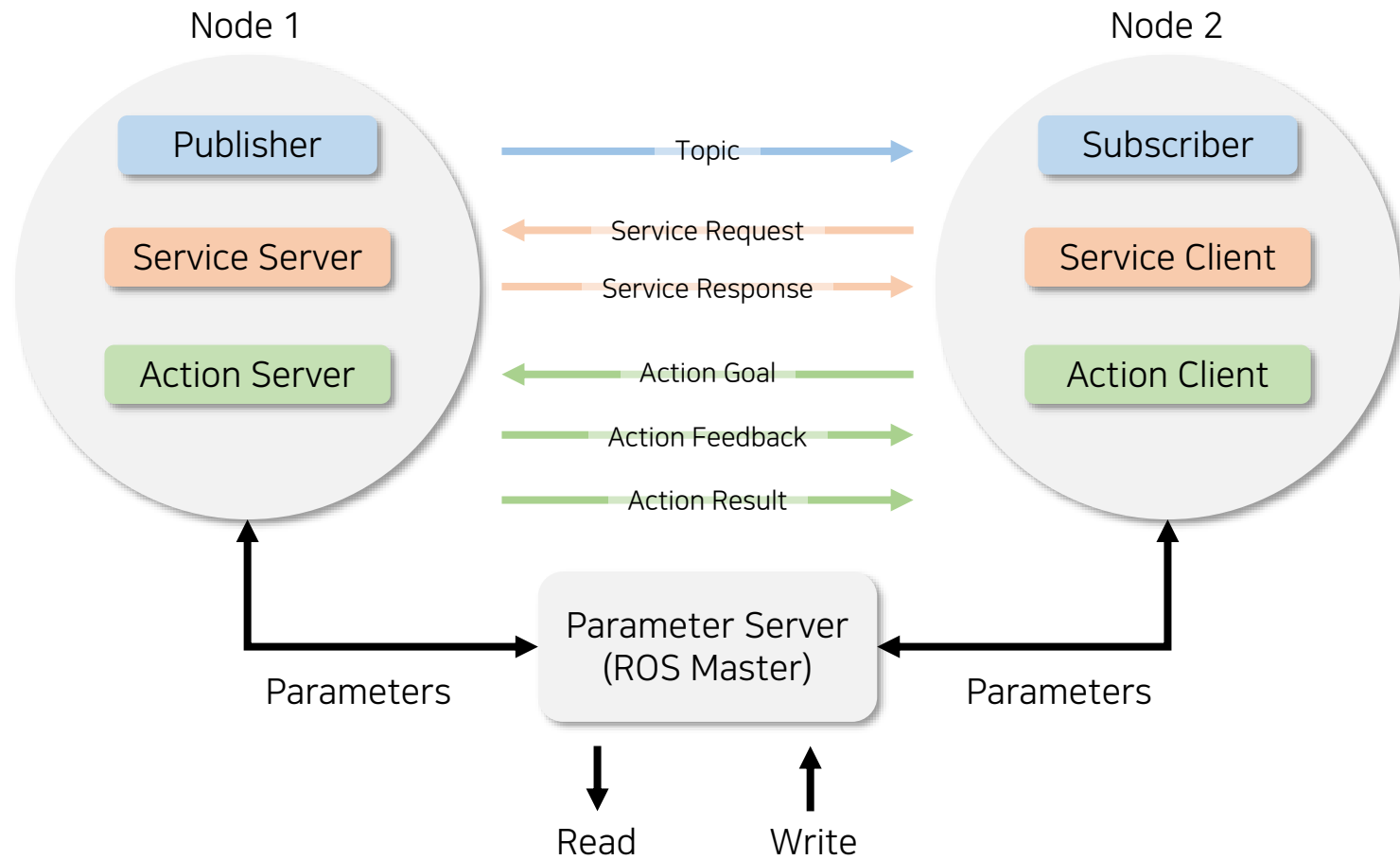
Compatible with different OS



# Making nodes and connecting them

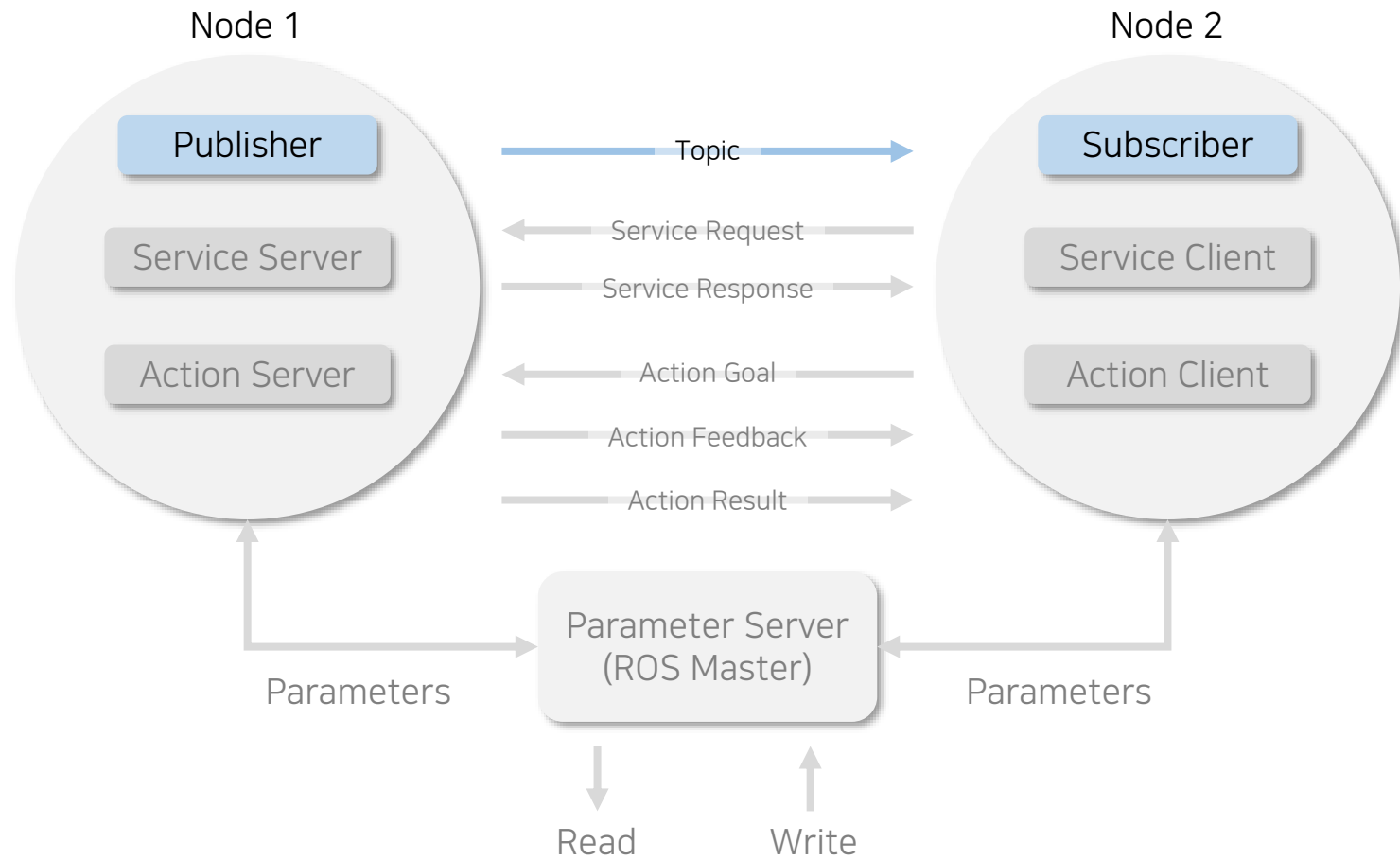
## Communications between nodes & ROS master

Chapter 7 : [https://github.com/robotpilot/ros-seminar/blob/master/07\\_ROS\\_기본\\_프로그래밍.pdf](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
~$ mkdir catkin_make
~$ cd catkin_make
~$ mkdir src
~$ catkin_make init
~$ echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
~$ source ~/.bashrc
```

- Goes to home directory
- Make a folder named "catkin\_ws"
- Goes inside the folder "capstone\_simulation"
- Important! : the folder name should be "src"
- Initialize the workspace
- Include the bash file in workspace
- Source ~/.bashrc

Now, let's create a package named "ros\_tutorials\_topic"

You are now at ~/catkin\_ws

```
~$ cd 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  
~$ ls
```

- Goes to ros\_tutorials\_topic
- List segments

You will see :

```
include  
src  
CMakeLists.txt  
package.xml
```

- Folder containing header files
- Folder containing source codes
- File for build settings
- 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

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

CMakeLists.txt

```
## 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})
```

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

```
time stamp
```

- time message named 'stamp'

MsgTutorial.msg

```
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 ..  
~$ cd src  
~$ gedit topic_publisher.cpp
```

- Goes back to upper directory
- Create a cpp file named topic\_publisher

```
#include "ros/ros.h"  
#include "ros_tutorials_topic/MsgTutorial.h"  
int main(int argc, char **argv)  
{  
    ros::init(argc, argv, "topic_publisher");  
    ros::NodeHandle nh;  
  
    ros::Publisher ros_tutorial_pub = nh.advertise<ros_tutorials_topic::MsgTutorial>("ros_tutorial_msg", 100);  
  
    ros::Rate loop_rate(10);  
    ros_tutorials_topic::MsgTutorial msg;  
    int count = 0;  
  
    while(ros::ok())  
    {  
        ***  
    }
```

- Basic ROS header file topic\_publisher.cpp
- MsgTutorial header file
- Main function of the node
- Initialize the node name as “topic\_publisher”
- Declare the node handle for communication with ROS system
- Declare the publisher ‘ros\_tutorial\_pub’ :
  - We will use MsgTutorial message
  - Publisher queue size is 100
- Set the loop rate as 10 : 10Hz
- Declare a message variable msg
- Initialize the variable ‘count’
- 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();  
    msg.data = count;  
    ROS_INFO("send msg = %d", msg.stamp.sec);  
    ROS_INFO("send msg = %d", msg.stamp.nsec);  
    ROS_INFO("send msg = %d", msg.data);  
    ros_tutorial_pub.publish(msg);  
    loop_rate.sleep();  
    ++count;  
}  
return 0;  
}
```

- Set the message 'stamp' as current time
- Set the message 'data' as 'count'
- Print out stamp.sec
- Print out stamp.nsec
- Print out data
- Publish the message
- Goes into sleep regarding to the loop rate
- Add 1 to variable 'count'

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)
```

[2]

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

void msgCallback(const ros_tutorials_topic::MsgTutorial::ConstPtr& msg)
{
    ROS_INFO("recievemsg= %d", msg->stamp.sec);
    ROS_INFO("recievemsg= %d", msg->stamp.nsec);
    ROS_INFO("recievemsg= %d", msg->data);
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "topic_subscriber");
    ros::NodeHandle nh;
    ros::Subscriber ros_tutorial_sub= nh.subscribe("ros_tutorial_msg", 100, msgCallback);
    ros::spin();
    return 0;
}
```

- Callback function which will run when message is received
- Main function of the node
- Initialize the node name as 'topic\_subscriber'
- Declare node handle
- Declare the subscriber 'ros\_tutorial\_sub':
  - We will use MsgTutorial message
  - Subscriber queue size is 100

Save & exit

# Making nodes and connecting them

Build the package & run

```
~$ cd ~/catkin_ws  
~$ catkin_make
```

- Goes to catkin\_ws directory
- Execute catkin build

On one terminal

```
~$ roscore
```

- Run ROS Master, parameter server, and roscore 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

The screenshot displays three terminal windows and the rqt\_graph interface. The leftmost terminal, titled 'roscore http://WAKANDA:11311/ 78x24', shows the output of 'roscore' and 'roslaunch' commands, indicating that the ROS master is running. The middle terminal, titled 'dongha@WAKANDA: ~ 75x24', shows the output of the 'topic\_publisher' node, which is sending messages to the '/ros\_tutorial\_msg' topic. The rightmost terminal, titled 'dongha@WAKANDA: ~ 74x24', shows the output of the 'topic\_subscriber' node, which is receiving messages from the '/ros\_tutorial\_msg' topic. Below the terminals, the rqt\_graph interface is shown, displaying a node graph with two nodes: '/topic\_publisher' and '/topic\_subscriber'. An arrow points from '/topic\_publisher' to '/topic\_subscriber', indicating the flow of data. The rqt\_graph interface also shows various settings and filters.

```
roscore http://WAKANDA:11311/ 78x24
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://WAKANDA:37897/
ros_comm version 1.12.14

SUMMARY

PARAMETERS
 * /roscpp: kinetic
 * /rosversion: 1.12.14

NODES
auto-starting new master
process[master]: started with pid [26812]
ROS_MASTER_URI=http://WAKANDA:11311/

setting /run_id to 0d72e8c8-c196-11e8-9325-4cedfb992114
process[roscpp-1]: started with pid [26826]
started core service [/roscpp]
```

```
dongha@WAKANDA: ~ 75x24
INFO [1537971166.426043514]: send msg = 425936454
INFO [1537971166.426043514]: send msg = 699
INFO [1537971166.526006352]: send msg = 1537971166
INFO [1537971166.526006352]: send msg = 525948557
INFO [1537971166.526125053]: send msg = 700
INFO [1537971166.625998759]: send msg = 1537971166
INFO [1537971166.626024284]: send msg = 625976208
INFO [1537971166.626033434]: send msg = 701
INFO [1537971166.726008207]: send msg = 1537971166
INFO [1537971166.726008207]: send msg = 725946373
INFO [1537971166.726130516]: send msg = 702
INFO [1537971166.825980142]: send msg = 1537971166
INFO [1537971166.826013868]: send msg = 825950940
INFO [1537971166.826026781]: send msg = 703
INFO [1537971166.926012144]: send msg = 1537971166
INFO [1537971166.926095677]: send msg = 925951075
INFO [1537971166.926133020]: send msg = 704
INFO [1537971167.025964762]: send msg = 1537971167
INFO [1537971167.026001925]: send msg = 25931843
INFO [1537971167.026012029]: send msg = 705
INFO [1537971167.125971625]: send msg = 1537971167
INFO [1537971167.126019709]: send msg = 125933369
INFO [1537971167.126038134]: send msg = 706
```

```
dongha@WAKANDA: ~ 74x24
INFO [1537971166.426460631]: receive msg = 425936454
INFO [1537971166.426478918]: receive msg = 699
INFO [1537971166.526543303]: receive msg = 1537971166
INFO [1537971166.526615886]: receive msg = 525948557
INFO [1537971166.526646419]: receive msg = 700
INFO [1537971166.626231865]: receive msg = 1537971166
INFO [1537971166.626250310]: receive msg = 625976208
INFO [1537971166.626256947]: receive msg = 701
INFO [1537971166.726576045]: receive msg = 1537971166
INFO [1537971166.726671229]: receive msg = 725946373
INFO [1537971166.726709767]: receive msg = 702
INFO [1537971166.826265265]: receive msg = 1537971166
INFO [1537971166.826290847]: receive msg = 825950940
INFO [1537971166.826300600]: receive msg = 703
INFO [1537971166.926583939]: receive msg = 1537971166
INFO [1537971166.926654668]: receive msg = 925951075
INFO [1537971166.926684379]: receive msg = 704
INFO [1537971167.026282973]: receive msg = 1537971167
INFO [1537971167.026286803]: receive msg = 25931843
INFO [1537971167.026296781]: receive msg = 705
INFO [1537971167.126336241]: receive msg = 1537971167
INFO [1537971167.126390373]: receive msg = 125933369
INFO [1537971167.126422457]: receive msg = 706
```

rqt\_graph

```
graph LR
    /topic_publisher -- "/ros_tutorial_msg" --> /topic_subscriber
```

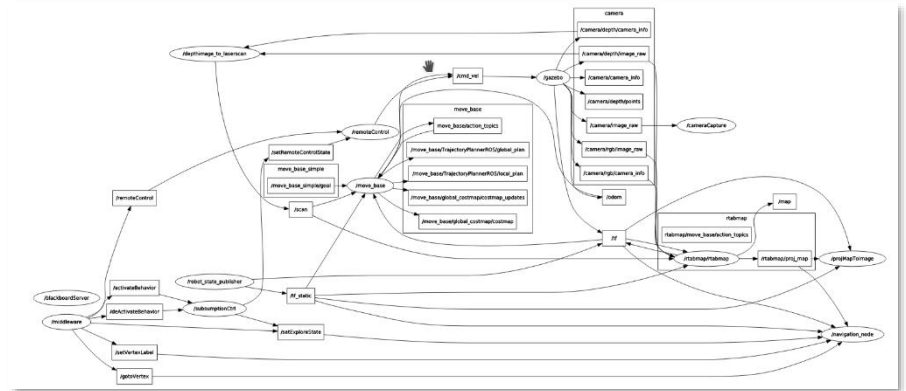
# Making nodes and connecting them

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

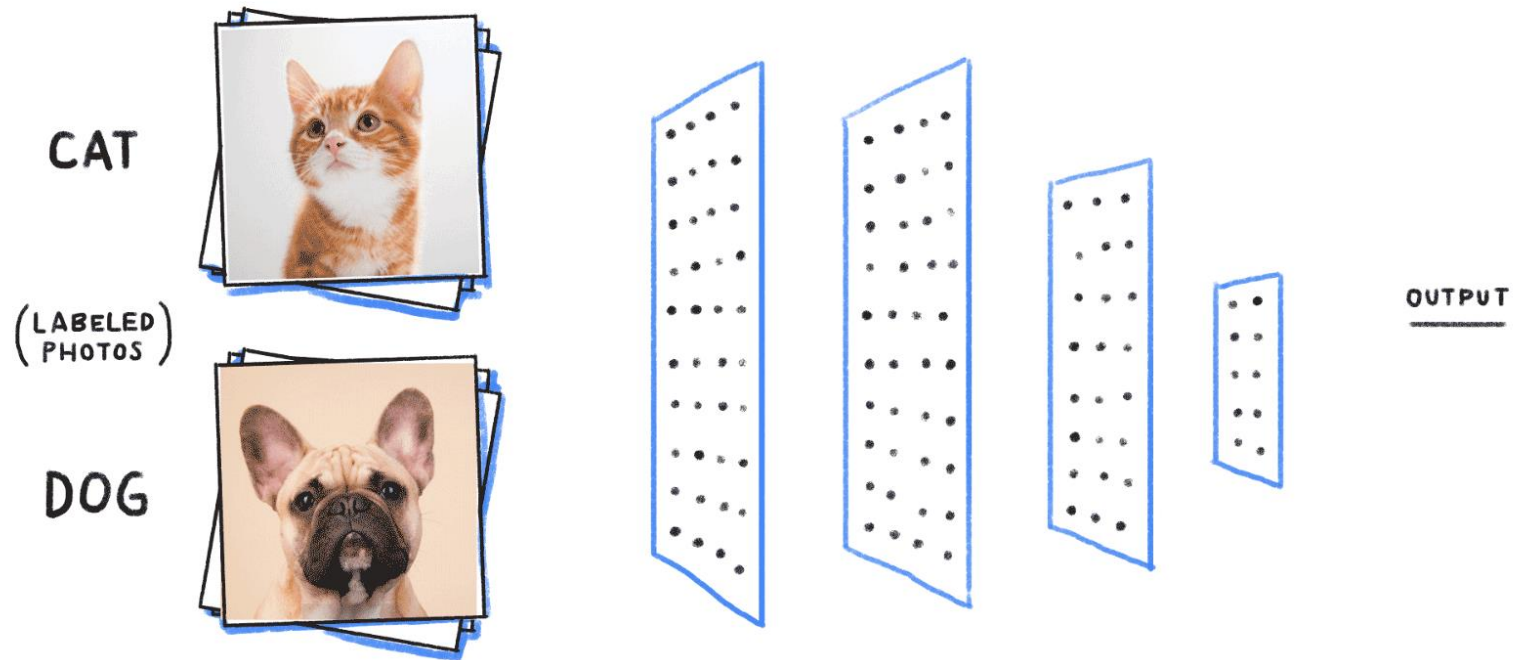
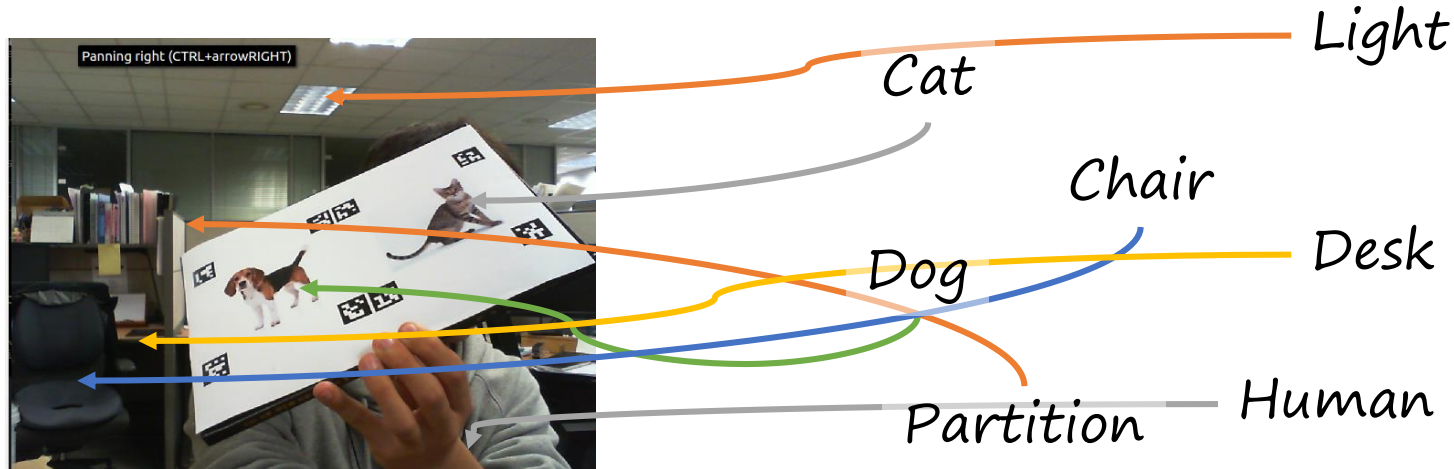


Image from <https://becominghuman.ai/building-an-image-classifier-using-deep-learning-in-python-totally-from-a-beginners-perspective-be8dbaf22dd8>

# 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>

# Getting the cat image and dog image from webcam

From webcam to readmarker

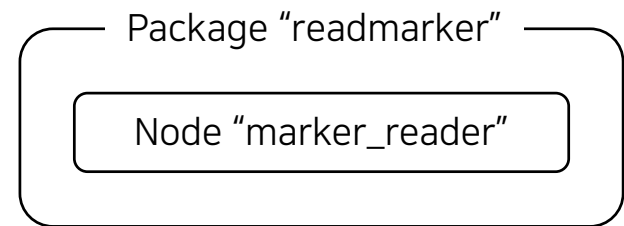
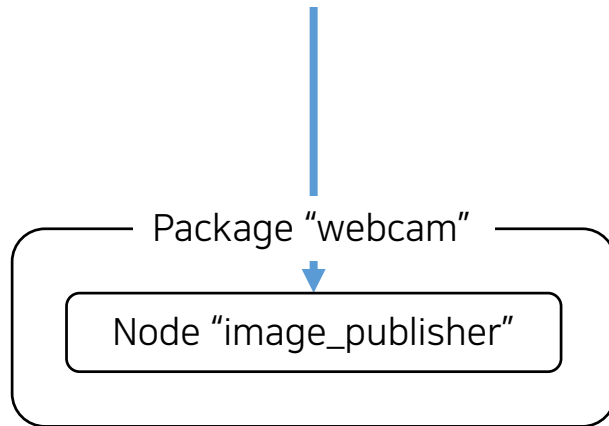


Capture a video frame from webcam 0 (if it does not work, try 1, 2, 3...)

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

Convert the image into Mat format

```
cap >> frame;
```



# Getting the cat image and dog image from webcam

From webcam to readmarker

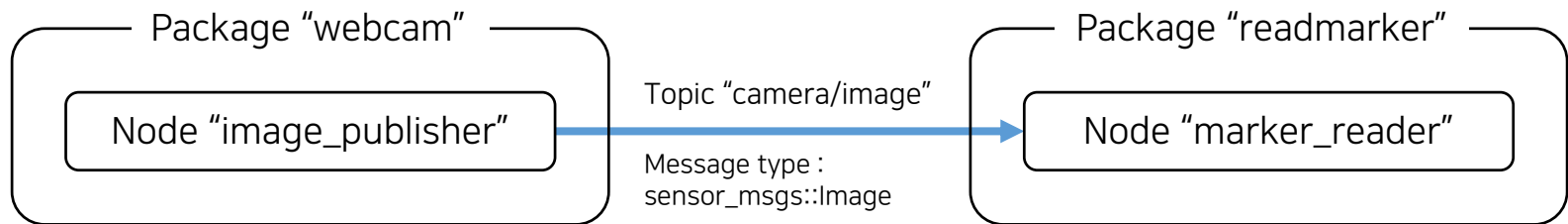


Convert the OpenCV Mat format to ROS message

```
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

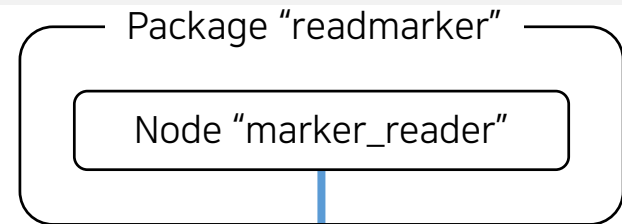
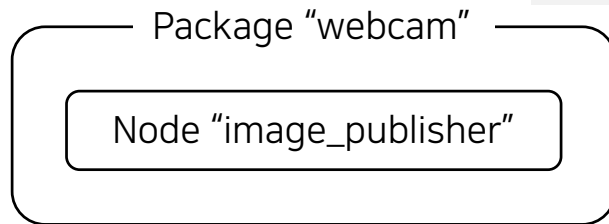
```
cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::image_encodings::BGR8);
```

Convert the OpenCV CvImage to Mat

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

Detect markers, crop, warp, & publish

...



Topic "cropped\_img"

Message type :

1. int32
2. sensor\_msgs::Image
3. sensor\_msgs::CompressedImage

Out



# Getting the cat image and dog image from webcam

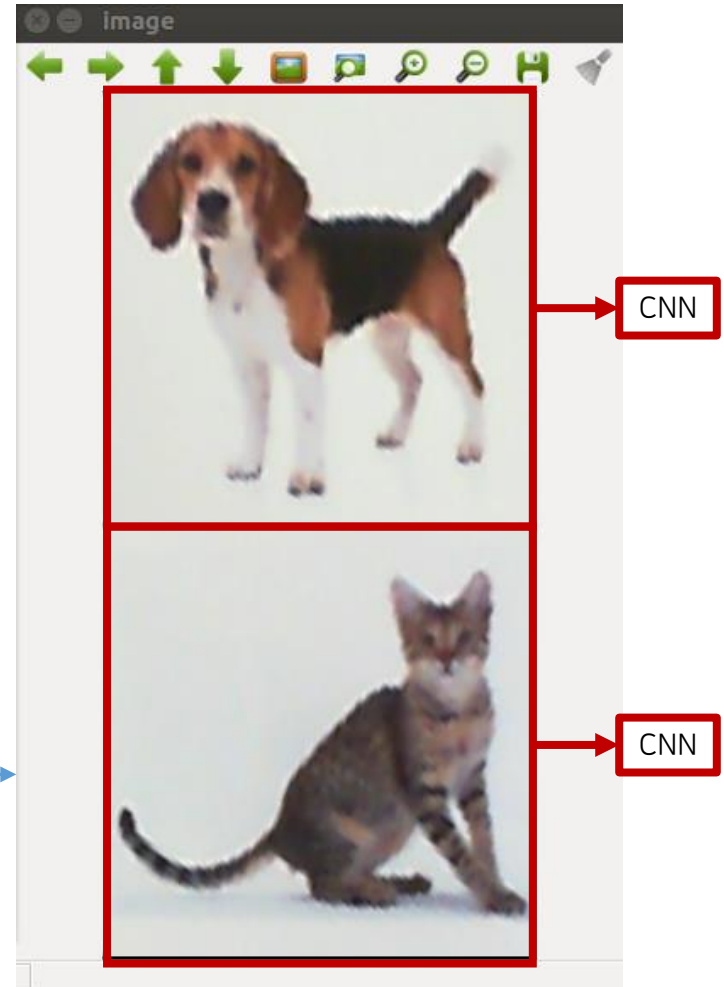
Use the Aruco markers to crop the images



Marker Detection

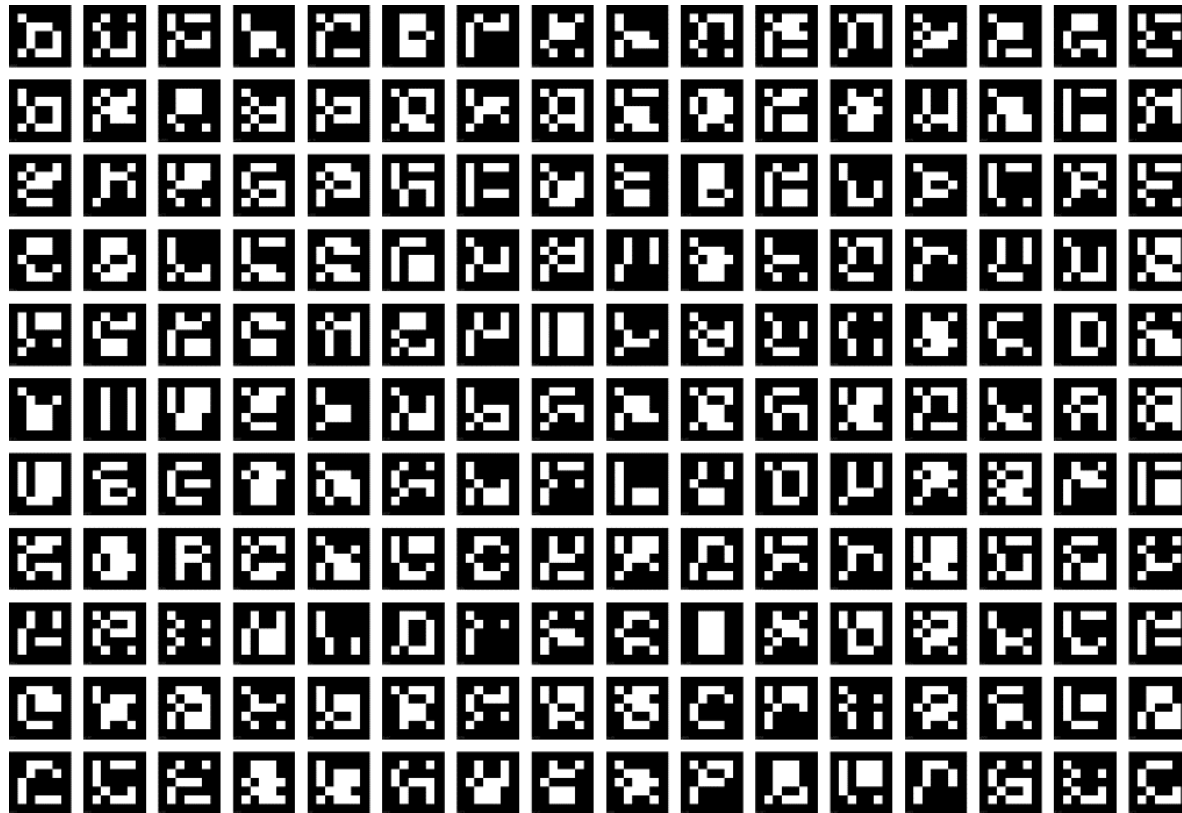


Get Individual Images



# Getting the cat image and dog image from webcam

AruCo in OpenCV



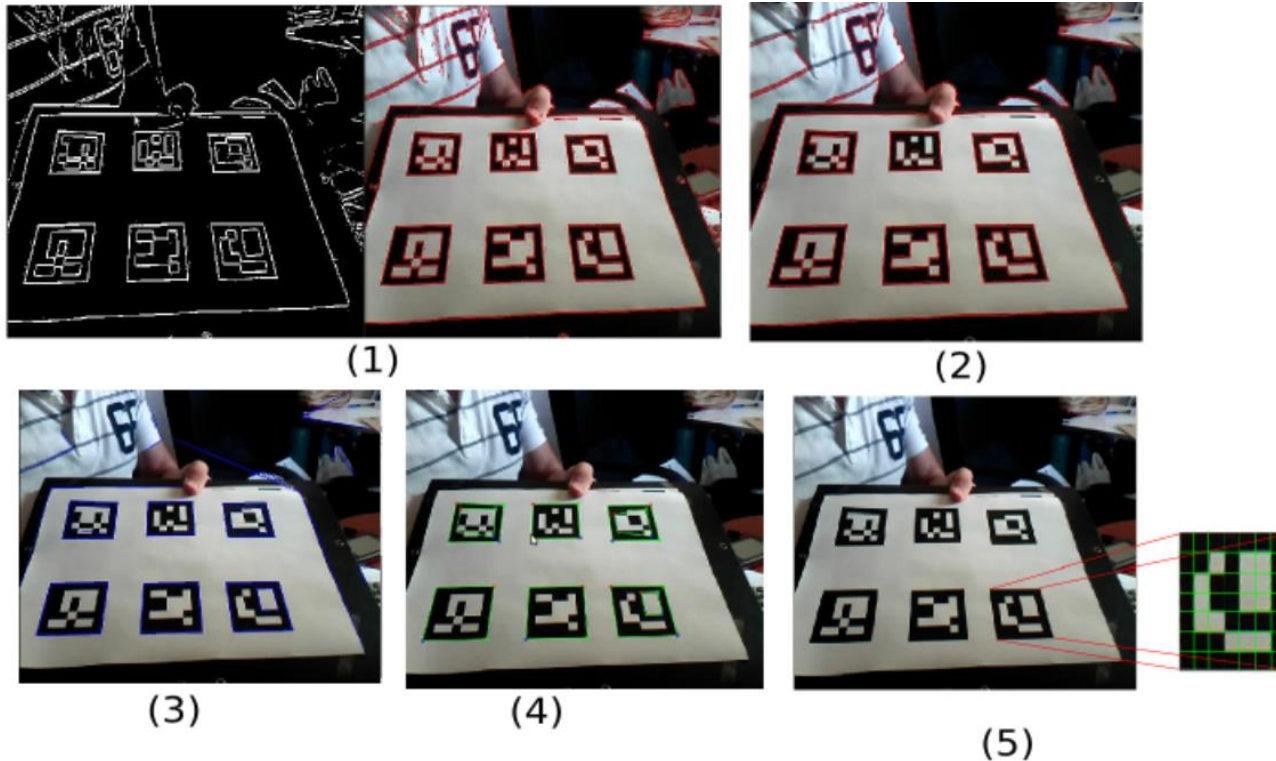
AruCo markers

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 a 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/1QU9KoBtjSM2kF6IT0jQ76xqL7H0TEtXriJX5kwi9Kgc/edit#>

# Getting the cat image and dog image from webcam

Use the Aruco markers to crop the images



Marker Detection



Detect the markers

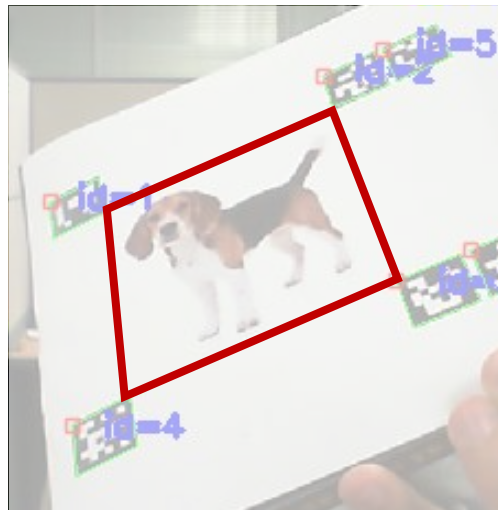
```
aruco::detectMarkers(inputImage, dictionary, markerCorners, markerIds)
```

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

b\_image1 and b\_image2 is used to check if the images are available

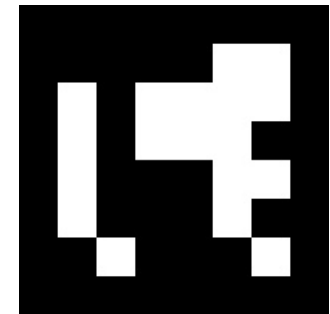
If image 1 (marker 1~4) is available : b\_image1 = 1,  
Get the coordinates of image 1 coordinates and find homography

```
if(b_image1==1){...}
```



markerCorners[i][0]

markerCorners[i][1]



markerCorners[i][3]

markerCorners[i][2]

# Getting the cat image and dog image from webcam

Use the Aruco markers to crop the images



Marker Detection



Detect the markers

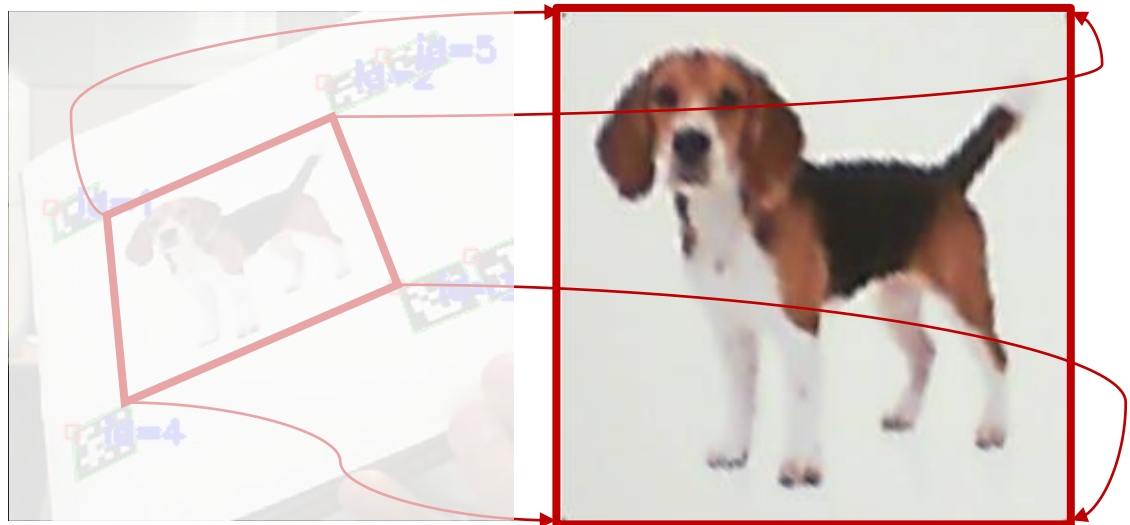
```
aruco::detectMarkers(inputImage, dictionary, markerCorners, markerIds)
```

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){...}
```

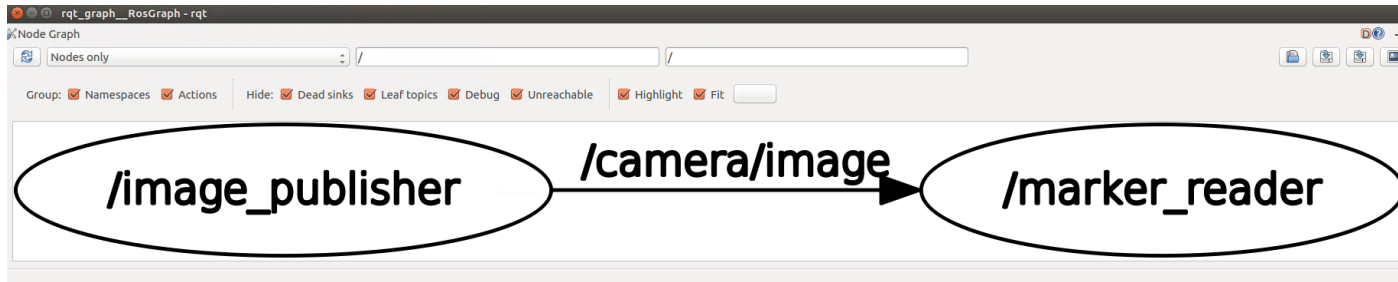


Homography (transformation matrix) :  $\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = H \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$

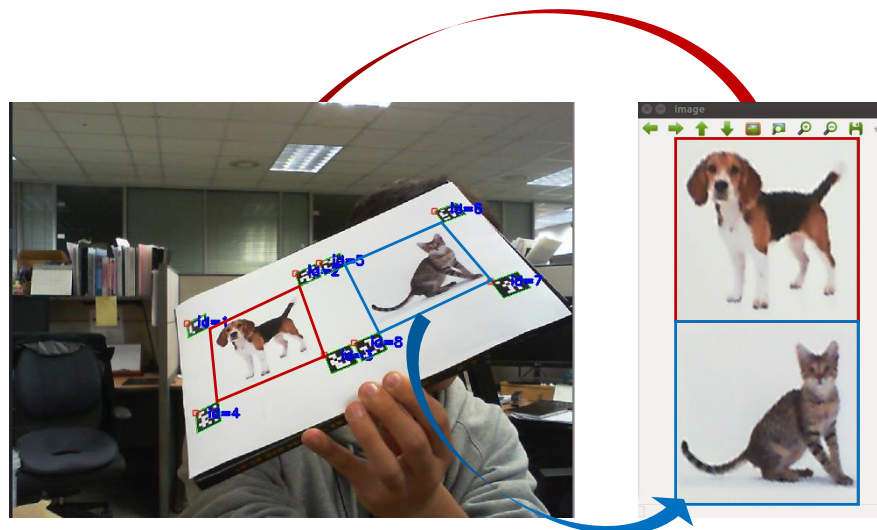


# In this assignment

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