# Robotics and ROS

Khan Saad Bin Hasan

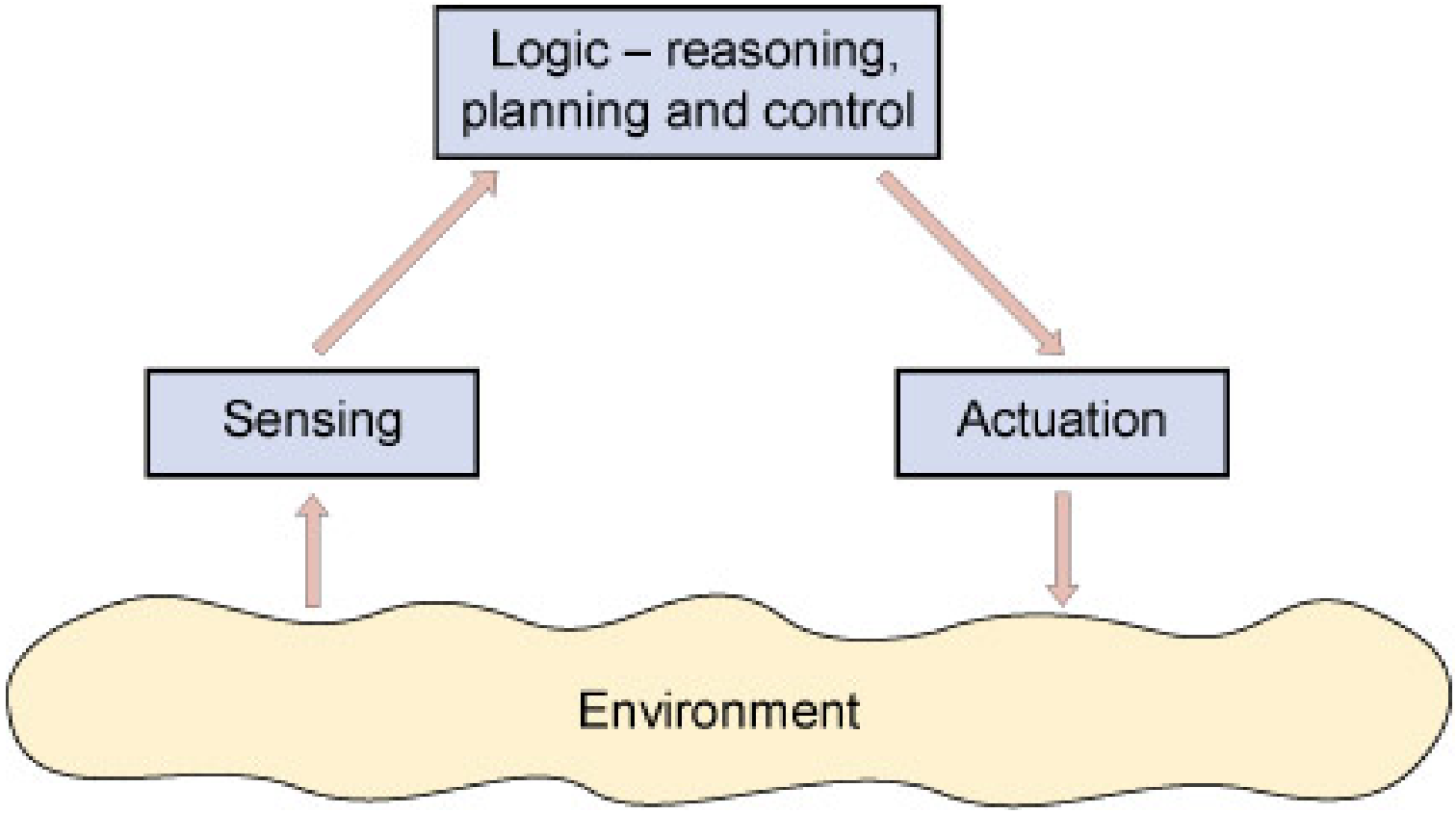
khansaadbinhasan.github.io

*khansaadbinhasan@gmail.com*

October 19, 2019

# Contents

# General Outline of a Robotic System



# Sensors and Actuators

- Sensors can never be perfect.
- Built up of a number of modules, Hence multiple points of failures e.g, LIDARs have a rotating top, a laser sender, a receiver and a PCB.
- Coordination between all the modules may never be perfect.
- The better a sensor is, the costlier it gets. e.g, IMUs, Cameras etc.

# Sensors and Actuators

- Even if a sensor is perfect, the state of the environment may depend on an extremely large number of variables.
- If we take all these variables into consideration, the computation may be infeasible.
- Actuators may make the sensors unreliable. e.g, not having a still camera while the camera assumes that it is at rest.
- Similar arguments may apply for actuators.

# Logic- Reasoning, Planning and Control

- Since, we don't get perfect data from sensors, We cannot have absolute algorithms i.e, algorithms that can determine the absolute state of the environment.
- We make use of probabilistic techniques for state estimation of Robot.
- These techniques do not guarantee correct solution.
- They provide a range of correct solutions, or a domain of possibilities while indicating the most likely solution.
- These include techniques like robot localization, SLAM, particle filtering etc.

# Collaboration among different parts

- A robot may consists of dozens of sensors and actuators and a number of computers.
- e.g, Humanoid robots may have dozens of servo motors in their hands and feet, many cameras for vision, IMUs for localization, Infrared and ultrasonic sensors for emergency safety etc.
- These sensors need some way to access the computational resources and actuators need to get commands from the decision process run on the computers. The computers may also need to communicate within themselves.

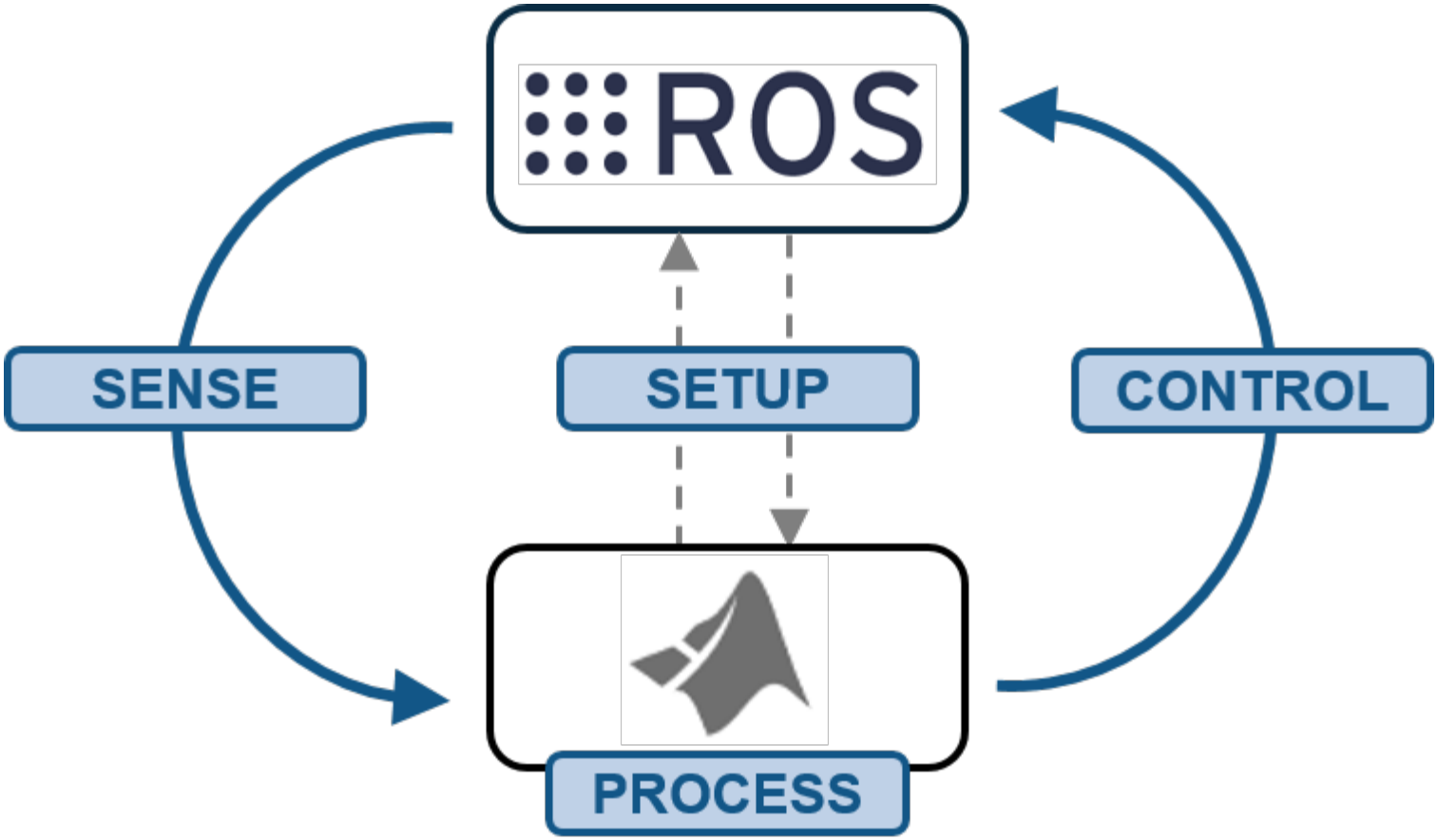
# Collaboration among different parts

- We cannot have these communications haphazardly, hence we need to have some guidelines and preferably some implemented protocols, packages etc. to handle these for us.
- Also, there are many algorithms such as Kalman Filter, PID Control, AMCL etc. which are often needed. Hence, it would make production faster if they were already implemented and in a uniform way i.e, following certain practices.
- We may also need to share code between different teams, it would be better if they follow the same conventions to easily share code.

# The Frameworks

- To address the above problems, a number of frameworks had been developed.[1]
- These frameworks were made for particular projects and could not be reused by others and the good ones were proprietary.
- Hence, there was a need of a middleware(or framework) that will act as the standard.
- ROS, which was also started to solve specific problems came up as the solution due to its open and flexible nature.

# What is ROS?



# Definition

## ROS Definition

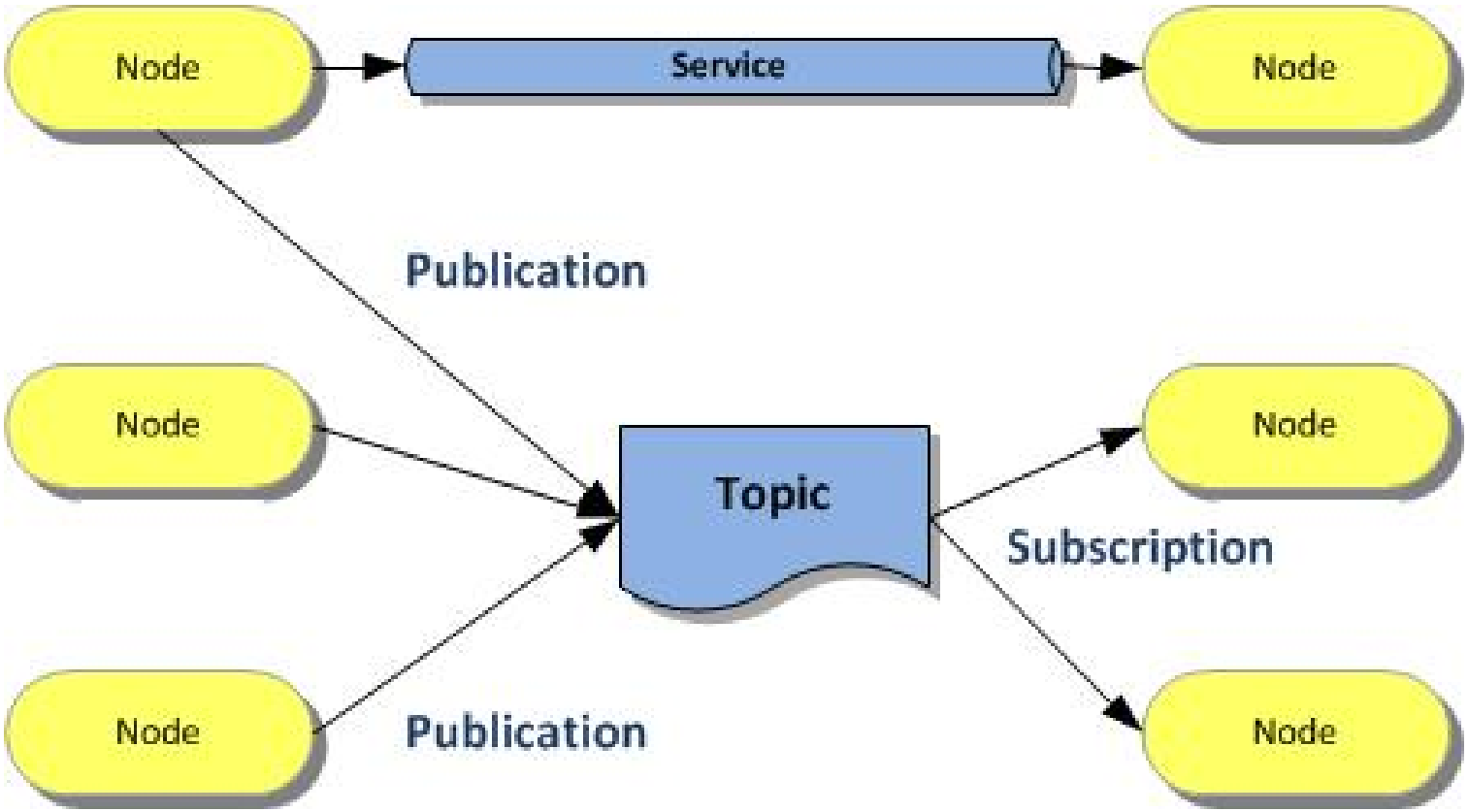
ROS, an open-source robot operating system. ROS is not an operating system in the traditional sense of process management and scheduling; rather, it provides a structured communications layer above the host operating systems of a heterogeneous compute cluster.[2]

- ROS is a middleware, a meta-operating system.
- A meta operating system works in close proximity with the operating system but does not provide all the functionality to be classified as operating system.

# The Operating System

- ROS can be run on any OS with hacks, but it runs better with linux operating system especially Debian based operating systems.
- The most convenient and stable configuration perhaps, in my opinion is ROS kinetic with Ubuntu 16.04
- A good knowledge of Linux is needed in order to work with ROS. e.g, how package management works in linux, the make system in linux, shell scripts etc.

# The Building Blocks



# Packages

## Packages

Software in ROS is organized in packages. A package might contain ROS nodes, a ROS-independent library, a dataset, configuration files, a third-party piece of software, or anything else that logically constitutes a useful module. The goal of these packages is to provide this useful functionality in an easy-to-consume manner so that software can be easily reused. In general, ROS packages follow a "Goldilocks" principle: enough functionality to be useful, but not too much that the package is heavyweight and difficult to use from other software.[3]

# Packages

Some of the important files/directories inside Packages are:

- Nodes: A node is a process that performs computation.
- CMakeLists.txt: It is the input to the CMake build system for building software packages.
- Package.xml : It defines properties about the package such as the package name, version numbers, authors, maintainers, and dependencies on other catkin packages.
- .yaml files: To run a rosnode you may require a lot of parameters e.g, Kp,Ki,Kd parameters in PID control. We can configure these using YAML files.
- launch files: To run multiple nodes at once in ROS we use launch files.

# Workspaces

## Catkin Workspaces

A catkin workspace is a folder where you modify, build, and install catkin packages. It can contain up to four different spaces which each serve a different role in the software development process.[4]

- The source space contains the source code of catkin packages. This is where you can extract/checkout/clone source code for the packages you want to build. Each folder within the source space contains one or more catkin packages.

# Workspaces

- The build space is where CMake is invoked to build the catkin packages in the source space. CMake and catkin keep their cache information and other intermediate files here.
- The development space (or devel space) is where built targets are placed prior to being installed. The way targets are organized in the devel space is the same as their layout when they are installed. This provides a useful testing and development environment which does not require invoking the installation step.
- Once targets are built, they can be installed into the install space by invoking the install target, usually with make install.

# Communication

## ROS Master

The ROS Master provides naming and registration services to the rest of the nodes in the ROS system. It tracks publishers and subscribers to topics as well as services. The role of the Master is to enable individual ROS nodes to locate one another. Once these nodes have located each other they communicate with each other peer-to-peer.[5]

# Communication

## Topics and Services

Topics are named buses over which nodes exchange messages. Topics have anonymous publish/subscribe semantics, which decouples the production of information from its consumption. In general, nodes are not aware of who they are communicating with. Instead, nodes that are interested in data subscribe to the relevant topic; nodes that generate data publish to the relevant topic. There can be multiple publishers and subscribers to a topic.[6]

## Actionlib

The actionlib package provides tools to create servers that execute long-running goals that can be preempted. It also provides a client interface in order to send requests to the server.[7]

# Tools

- Robot Development requires a large amount of tools for debugging and to visualize the state of sensors and actuators.
- Simulations are also required in order to apply reinforcement learning and for faster, cheaper and safer testing.
- ROS provides a wide range of such tools and allow customization and to build your own tools.
- Examples include Gazebo for simulation, Rviz for sensor and actuator visualization, rqt tools for visualization of data etc.

# How to get started

- ROS maybe overwhelming for beginners to start.
- A good knowledge of computer engineering basics and hands on experience with linux operating system and atleast one popular programming language is helpful.
- Book recommendations- A Gentle Introduction To ROS[8], Programming Robots with Ros: A Practical Introduction To The Robot Operating System[9], ROS Robotics By Example[10]
- Please read the documentation at: wiki.ros.org and for queries visit: answers.ros.org .

# Thank you

James Kramer and Matthias Scheutz.
Development environments for autonomous mobile robots: A survey.
*Autonomous Robots*, 22(2):101–132, 2007.

Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng.
Ros: an open-source robot operating system.
In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

ROS Admin.
*Packages*, 2019 (accessed October 19, 2019).
http://wiki.ros.org/Packages.

ROS Admin.
*Workspaces*, 2019 (accessed October 19, 2019).
http://wiki.ros.org/catkin/workspaces#Catkin_Workspaces.

ROS Admin.
*Master*, 2019 (accessed October 19, 2019).
http://wiki.ros.org/Master.


ROS Admin.
*Topics*, 2019 (accessed October 19, 2019).
http://wiki.ros.org/Topics.


ROS Admin.
*Actionlib*, 2019 (accessed October 19, 2019).
http://wiki.ros.org/actionlib.


Jason M O'Kane.
A gentle introduction to ros.
2014.


Morgan Quigley, Brian Gerkey, and William D Smart.
*Programming Robots with ROS: a practical introduction to the Robot Operating System*.
" O'Reilly Media, Inc.", 2015.


Carol Fairchild and Thomas L Harman.
*ROS robotics by example*.
Packt Publishing Ltd, 2016.