

Jonathan Hanbali

**(a) The four main ecosystem components of a platform for personal Computer (PC), Personal Phone (PP) and Personal Robot (PR) respectively**

=======================================================================

The four main ecosystems are the **User, Service, Operating System, and Hardware**

**Personal Computer (PC)**

**User:** The user will be all the same (everyone or just people in essence)

**Service:** Virtual Box, Browsers (Google Chrome, Firefox), Team Viewer, Skype, Microsoft Office

**Operating System:** Windows 10, Windows 8, Linux, Fedora, Apple OS X, Ubuntu, Debian

**Hardware:** Gaming Computer, Laptop, Processor (i3, i5, i7, i9), Graphics Card, PSU

**Personal Phone (PP)**

**User:** The user will be all the same (everyone or just people in essence)

**Service:** Mobile Phone Apps (Browsers like safari), Chase mobile app, Venmo, Mobile phone games, Amazon app on mobile phone, etc

**Operating System:** Windows 8 Phone, Symbian, iOS, Android

**Hardware:** Phone (iPhone, Samsung Galaxy Phone, Blackberry phone, PDA)

**Personal Robot (PR)**

**User:** The user will be all the same (everyone or just people in essence), people that uses and design robots in particular

**Service:** Helping the user to move stuffs, Go to a certain distance, move objects, helping diffuse a bomb (work that’s hazardous for human)

**Operating System:** ROS, Galapagos , OPROS, NAOqi, fetch Robotics, RT middleware

**Hardware:** “Robots”, to be exact, servos, actuators, sensors like lidar, etc

=======================================================================

**(b) Robot software platform**

A robot software platform is a set of software tools and libraries designed to simplify the development of robotic applications. It includes middle-ware for communication (these middle-ware established an interface between the hardware platform and the software platform), libraries for sensor data processing and motion control, whereas programmer will be able to write a software without any knowledge of hardware and development environments for writing robot-specific software while focused on the service provided to the users.

=======================================================================

**(c) Robot Operating system (ROS) and software framework**

ROS is an **open-source, meta-operating system for your robot that will provide the service that you can expect from an Operating System (like Windows, or Ubuntu)**. They will include hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It would also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.

The software framework for developing the robot software will include **(plumbing, tools, capabilities, and ecosystems).**

- The software framework will help you to jointly develop complex programs by dividing the complex programs with messages exchanging method between nodes.

- It will support command tool, visualization tool Rviz, GUI toolbar rqt, 3D simulator Gazebo

- support modeling, sensing, recognition, navigation, and manipulation functions commonly used in robotics

- and finally, it will help you to create a robotics ecosystems.

=======================================================================

**(d) ROS configuration**

There are several “layers” of ROS Configuration

**Client Layer:** roscpp, rospy, roslisp, rosjava, roslibjs

**Robotics Application:** MoveIt!, navigation, rocon, teleop pkgs, descartes, people, ar track

**Robotics Application Framework:** tf, Robot locatlization, Image Pipeline, Ecto, Laser Filters, Ros Realtime, Mavros, Vision OpenCV

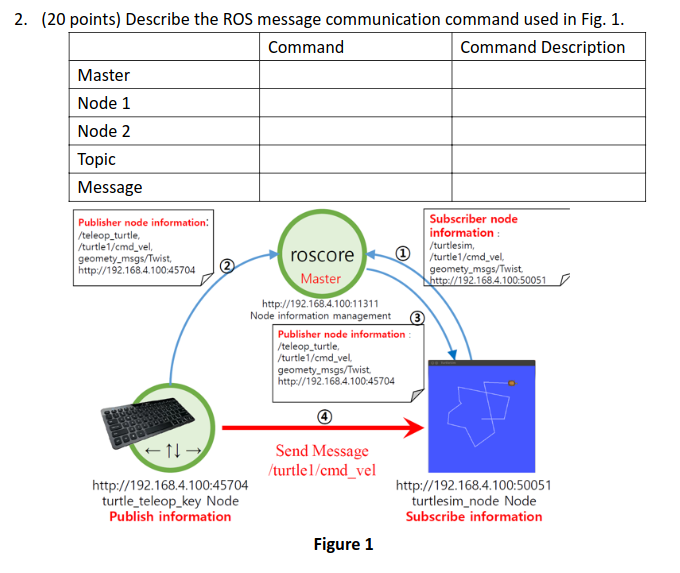
**Communication Layer:** Common msgs, rosbag, actionlib, pluginlib, rostopic, roservice, rosnode,

**Hardware Interface Layer:** Camera drivers, audio common, GPS/IMU drivers, Joystic drivers, rosserial, 3D sensor driver, diagnostic, ROS canopen

**Software Development Tools:** Rviz, rqt, wstool, rospac, catkin, colcon, rosdep

**Simulation:** Gazebo ROS pkgs, stage ROS

===========================================================================================================================================================================================================================



**Master:**

**Command:** ROSCORE

**Description:** This is the master “server” that will provide the Node information management. After we all the publisher node, the master will informs the subscriber node of the new publisher information.

**Node1:**

**Command:** turtlesim **turtlesim\_node**

**Description:** turtlesim will be the name of the package, and the turtlesim\_node will be the name of the node, node 1. This particular node will open a window or an “application window” where we can see the turtle that we can interact with. This node will also be the “information subscribe” node (subscriber node)

**Node2:**

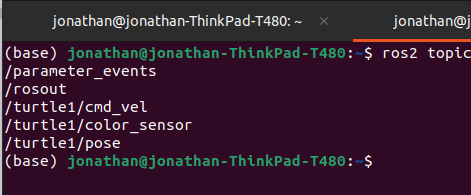
**Command:** turtlesim **turtle\_teleop\_key**

**Description: t**urtlesim is the name of the package, and the turtle\_teleop\_key is the name of node 2. This node will allow the user to send a “command” to the node 1 using keyboard to move the turtle into a certain direction. This node will be the “information publish” node (publisher node)

**Topic:**

**Command: ros2 topic list** (To show list of the available topic)

**Description:** Topic happens when the publisher node sends a message to the subscriber node. (w2 p16-17). In topic mode, messages are continuously transmitted unless the connection is terminated. So in essence, when we run teleop\_key, and we click the arrow or the shortcuts, we will send a topic to the subscriber node, that will be used to move the turtle on node 1 Topic is a part of messages, other than service, action, and parameter. To see what kind of topic you can use, you can use command ros2 topic list, and as you can see, we can use cmd\_vel, color\_sensor, pose, etc. And on the picture given at the top, there’s topic of: **/turtle1/cmd\_vel**



**Message:**

**Command: one of the example command is geometry\_msgs/Twist**

**Description:** messages is a type of data travel around nodes, such as topics, services, and actions. The type could be simple like integer, floating point, boolean, and it also have a simple data structure, perhaps also include an array of data structure in which message are listed. The message above will make the turtle “twist” or run a certain way depending on the value given by the users and the amount of iteration given.

===========================================================================================================================================================================================================================



Two types of sensors used in robotics and their capability:

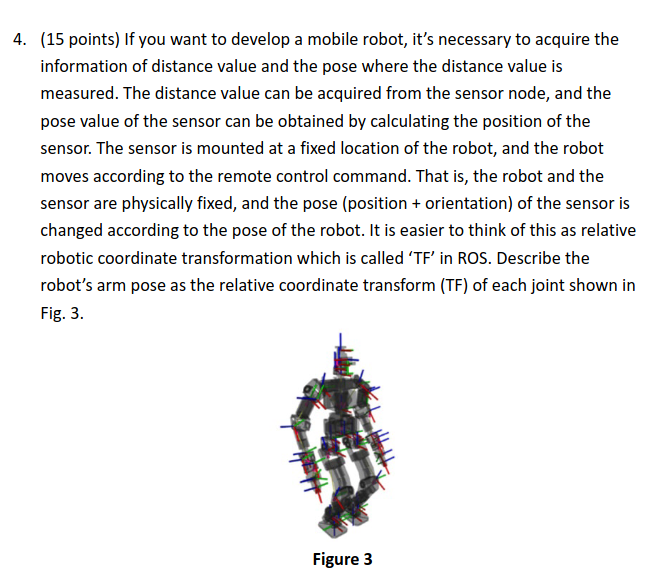
There’re several types of sensors in a sensor package, such as 1D, 2D, 3D range finders / sensors, Audio/Speech Recognition, Cameras, and Sensor Interface.

**Camera:**

Camera driver used for object recognition, face recognition, character recognition, and various application package. There are several different cameras that we could use such as stereo camera, and even a normal PC camera (USB camera). Camera will provide “vision” to our robots. With just a camera, we can perhaps include the object and face recognition to our robots, with more sensors such as LIDAR, and distance sensors, the camera can do so much more, and perhaps can help in creating the object / obstacle avoidance (by “fusion / combining multiple sensor data to create a better and more solid interpretation)

**3D sensors:**  
One of the best example for a 3D sensors is the IMU. IMU or an Inertial Measurement Unit is an example of the 3D sensors that measures and reports the body’s specific force, angular rates, and sometimes orientation using mostly accelerometer and gyroscope. Sometimes they include magnetometers as well. The IMU will provide the robot with a sensor information data such as yaw, pitch, and roll, when processed correctly. With those information, we can create a better movement because of the sensor reading (like a feedback to ensure that your robot moved to the right direction). IMU sensors usually read in all 3 axis (x, y, and z).

===========================================================================================================================================================================================================================



Distance value can be acquired from the sensor node

Pose value can be obtained by calculating the position of the sensor

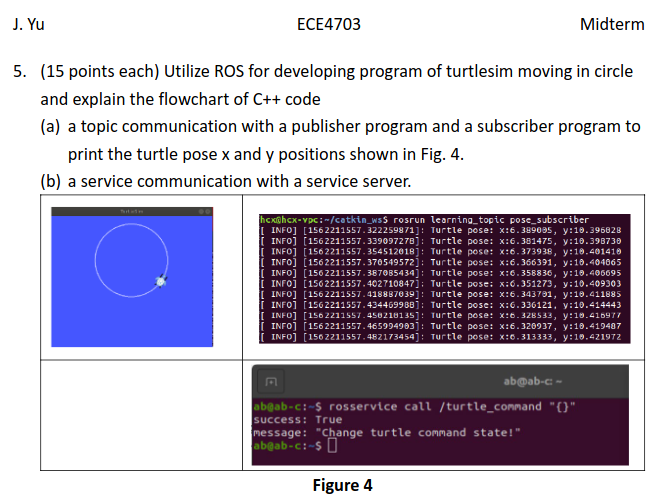
Sensor is mounted at a fix location of the robots, and the robot will moves according to the remote control command. **Position and the orientation of the sensor is changed according to the pose of the robots.** “Relative robotic coordinate transformation (TF)”. **Describe the robot’s arm pose as the relative coordinate transform (TF) of each joint shown.**

The TF will work as a “relative coordinate transform” of each joint which indicates the relationship between joints in the form of tree structure. TF is a powerful tool for managing the coordinate frames of different sensors and joints on a robot. It allows you to keep track of the relationships between different coordinate frames as they move and change orientation.

The base will be the coordinate 0,0,0 in (maybe the center), and then everything else will be in relation to or related to the base frame. There’re two movement that can happen in which is a translation, and a rotational movement. Like when we use twist message on the turtle, there’s a “distance” and “angular movement”. The translation is a “distance / coordinate” relative to the base frame, while the angular will be related to the rotational relative to the base.

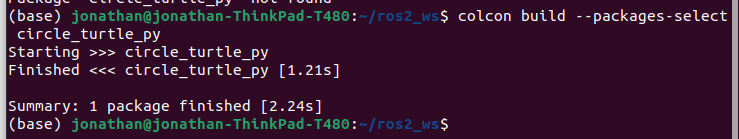
So let’s say in a robots, we will have that main coordinate or center frame, then on the base of the arm, it will be “joint 1”. This joint will have a relative coordinate with the base or the center, while let’s say “joint 2”, in which perhaps the “elbow” of the robot arms will have a different base frame (if the movement instead get affected by the other joint instead of the main joint), and this is why it is called as Relative Coordinate Transform since the “base” will be relative depending on which frame of reference you’re using. So it would be ideal to use a software than manually have to keep track of the frame of reference for each joint of the robot in which will be super complicated.

===========================================================================================================================================================================================================================

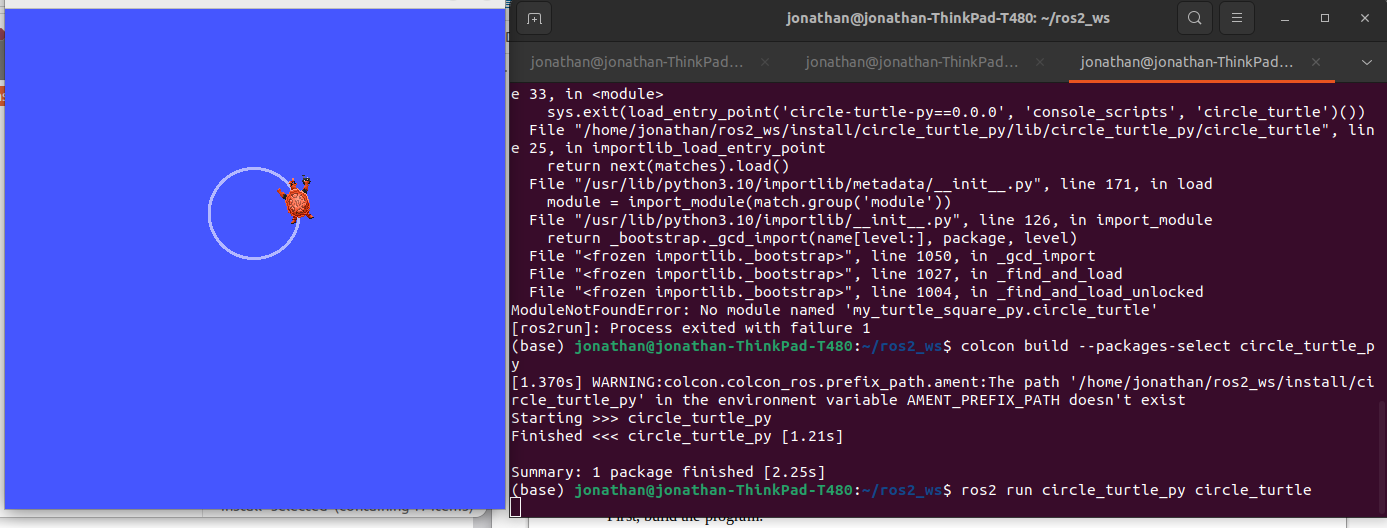


**(a) PUBLISHER AND SUBSCRIBER**

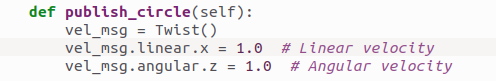
First, build the program:



Second, run it:

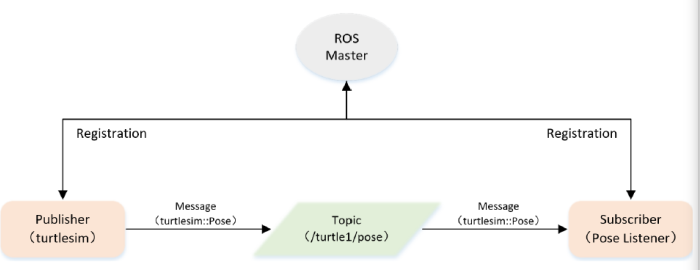


in the code we can see that:

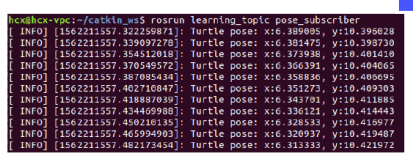


We set the linear movement on x (linear velocity on x) as 1.0, and angular movement on z axis (angular velocity on z) as 1.0. with this, we will be able to create a circle.

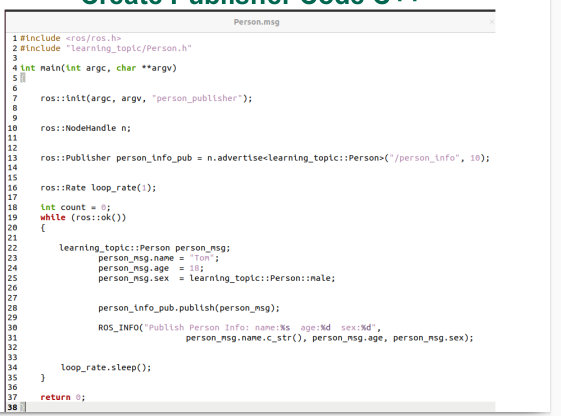
If we use publisher and subscriber, then the circle\_turtle program will run as the topic of the publisher program in which is the turtlesim.



As we can see from the flowchart above, the publisher will be the turtlesim program, the topic will be the pose (in this case is my circle\_turtle program in which will make the turtle turns in circle. If we want to create a subscriber, then we can listen from the topic about the linear and angular velocity of the turtle, then write it down on the subscriber node like given on the w5 note. So with a fully functioning subscriber program, we will have the terminal shows the coordinate of the turtle.



Another example of publisher and subscriber node is the person that we did



So we will init the program, then create a nodehandler n, we will then advertise the topic that we have the person info, then we create a loop. Then when the subscriber ask about the person\_msg, the person subscriber will give the information about the person in which is now contained:  
.name = Tom, .age = 10, .sex = male



On the subscriber node, now when we call the code, it will show the information using ROS\_INFO. It will get the message as a pointer, and then get the value of name, age, and sex from the message that’s being broadcasted by the publisher node. So the subscriber will subscribed to the broadcasted message from the person publisher node.