TCS Innovation Lab

Project Report

May 2017 - July 2017

Development of ROS Interface for TAL BRABO Robotic Manipulator

Student:
Sudhir PratapYadav
Shivam Jaiswal

Mentor:
Dr. Swagat Kumar
Head
Robotics Division
TCS Innovation Lab

September 8, 2017

1 Introduction

Tal Brabo is 5 DOF robotic arm developed at TAL pvt. Ltd. This document contains information about development of software interface (driver) for controlling robot using ROS. A usb camera is attached at end-effector of robot. Robot is required to follow a circular object.

2 Project Objective

This project is about controlling a 5-dof manipultor arm (TAL BRABO) via ROS. This involved following steps

- Controlling motors of robot using arduino.
- Developing ROS node (driver) to communicate with arduino based on joint position streaming.
- Designing pipeline for object follower using a USB camera attached to end-effector of robot to demonstrate working of complete system.

3 Preliminaries

3.1 ROS (Robot Operating System)

Robot Operating System(ROS) is a Robotics middleware(i.e. collection of software frameworks for robot software development). Even though ROS is not an operating system, it provides services designed for heterogeneous computer cluster such as hardware abstraction, low-level device control, implementation of commonly used functionality,message-passing between processes, and package management. Running sets of ROS-based processes are represented in a graph architecture where processing takes place in nodes that may receive, post and multiplex sensor, control, state, planning, actuator and other messages. [1]

Hardware Abstraction This means encapsulation of hardware details via standard functions to provide safe access to control any hardware. Driver is a piece of software used to provide software level control over hardware. This process eliminates the need to write routines which are dependent on hardware as one can use drivers built by users/community or company delivering the product. This concept is very helpful for researchers so that they can focus on their research algorithms instead of diving into hardware level programs.

Full-fledged discussion of ROS is out of scope of this document but we recommend reader to go through following articles [2], [3] and [4].

3.2 Timers and Interrupts in Arduino

Timer module in arduino or any micro controller is used to measure precise timing. This is required to generate precise pulse signals for controlling motors. An interrupt is an external event that interrupts the running program and runs a special interrupt

service routine (ISR) [5]. Interrupts are used for implementing parallel routines in micro controllers. In this project, Serial Interrupt is used to read command data without disturbing the pulse generation process. We assume reader has good knowledge about working or timers and interrupt in arduino. In case reader finds this concept bit shaky we recommend reader to read following article [5].

4 System Architecture for controlling Robot

Figure 1 shows architecture used to control TAL robot. Green box includes robot hardware to be controlled, while blue box represents hardware and software architecture designed by us to control robot. We chose to build a layer wise design for modularity and efficiency.

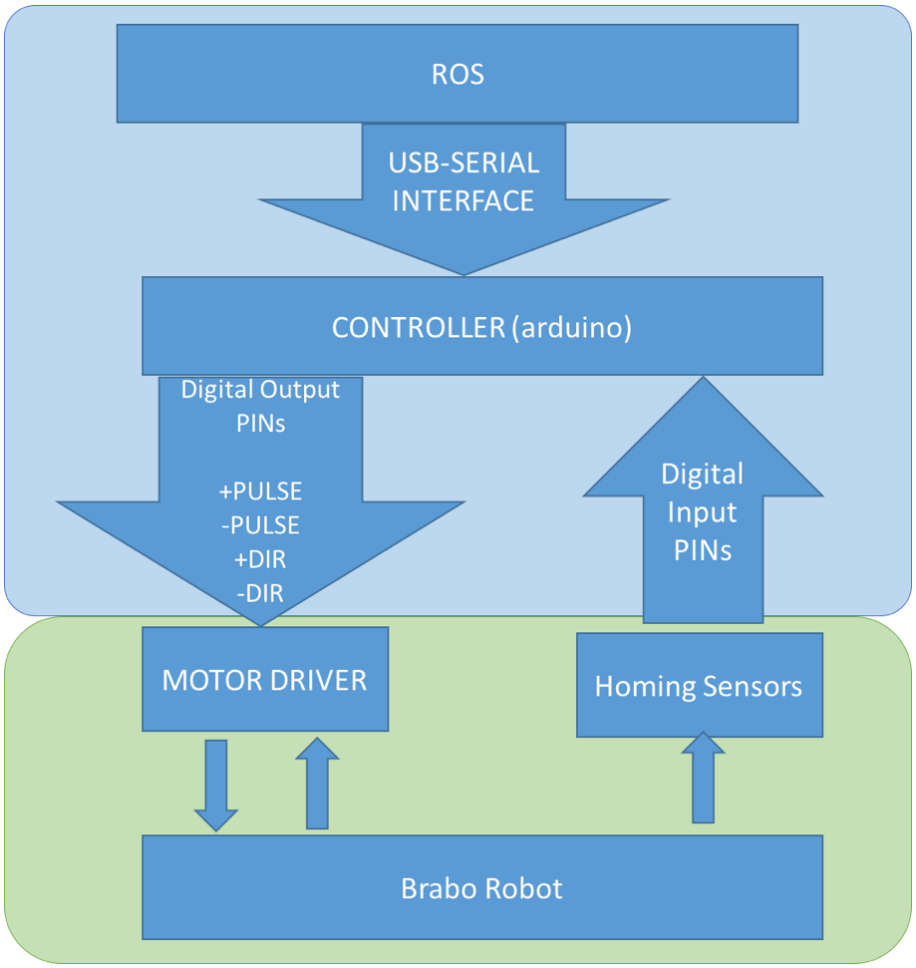


Figure 1: Layer wise system architecture for controlling TAL robot via ROS

4.1 Layer 2: TAL ROBOT Hardware Interface

TAL robot one of the procurement of TCS from TAL enterprises – the first and only manufacturer of industrial robot in India. It can perform various operations like Mig Welding, Pick and Place, Insert and Open, Copying the motion etc. The robotic hand with 5-axis of rotation and can carry a load of 10 kg. Base (1st Joint),

Shoulder (2nd Joint) and Arm (3rd Joint) are controlled by AC powered Motors and Drivers while Wrist (4th Joint) and Hand (5th Joint) are controlled by DC powered Motors and Drivers.

4.1.1 Motor Specifications

Motors installed in robot are step servo motors built by LEADSHINE company. 1st ,2nd and 3rd joints uses ES-MH series step servo motors with holding torque ranging from 8 Nm to 20 Nm. While 4th and 5th joint uses ES-M series step servo motors with Holding Torque ranging from 1 Nm to 8 Nm. Motor have two cables, one is encoder cable that is connected to the motor driver to provide feedback for working of internal PID control. While another is simply used to control motor. Motors also have a brake system which is activated when power is off to avoid falling of robot on ground. EB+ and EB- are brake connection. VCC and GND are fixed supply. EA+ and EA- are control input.

A: HDD15 Female Pin	Wire Color	B: HDD15 Male Pin	Name	Description
1	Black	1	EA+	Channel A+
2	Red	13	VCC	+5V power input
3	White	3	GND	+5V GND
11	Yellow	2	EB+	Channel B+
12	Green	12	EB-	Channel B-
13	Blue	11	EA-	Channel A-

Figure 2: Motor Wiring Details

4.1.2 Motor Driver Specification

Leadshine's ES-DH2306 and ES-DH1208 drives ES-MH series step servo motors are used for 1st 2nd and 3rd motors. While Leadshine's ES-D508, ES-D808 and ES-D1008 drives ES-M series step servo motors are used for 4th and 5th joints. The system includes an easy servo motor combined with a fully digital, high performance easy servo drive. The internal encoder in the motor is used to close the position, velocity and current loops in real time, just like servo systems. It combines the best of servo and stepper motor functions and increases the accuracy. Figure 3 shows control loop of motor driver.

4.1.3 Control Signal required by Motor Driver

Motor driver require pulse and direction signal for moving motor. Motor move one step with one pulse based on direction pulse. To make a reliable operation, the ES drive requires the control signals to meet the setup time requirements as shown in figure 4. Otherwise losing of steps may happen. Motor driver can be connected by a male 44pin D-sub connector. To control motor it only requires 4 pins shown in figure 5.

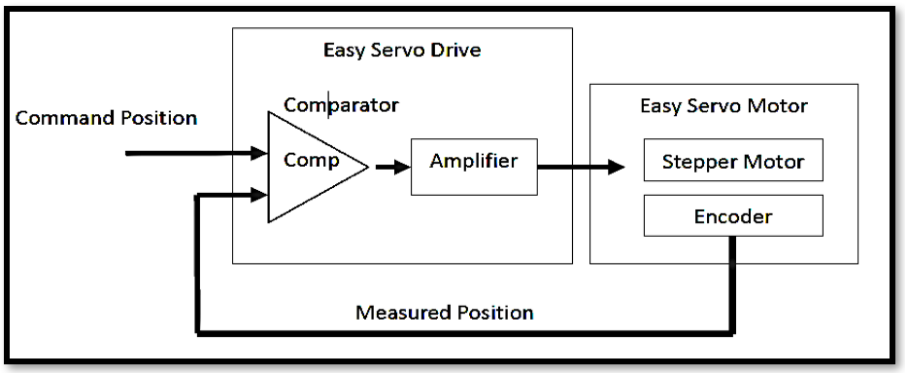
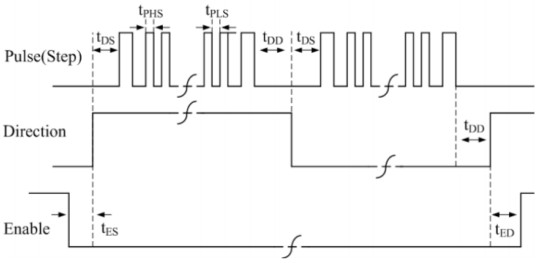


Figure 3: Control loop inside motor driver



Symbol	Description
t_{DS}	Direction Setup Time
t_{PHS}	Pulse High Level Setup Time
t_{PLS}	Pulse Low Level Setup Time
t_{DD}	Direction Delay Time
t_{ES}	Enable Setup Time
t_{ED}	Enable Delay Time

Control Signal Setup Time						
Drive	Frequency	tDS	tPHS / tPLS	tDD	tES	tED
ES-DH2306	200K	>5uS	>2.5us	>5uS	>50ms	>50ms
ES-DH1208	200K	>5uS	>2.5us	>5uS	>50ms	>50ms

Figure 4: Control Signal Setup Timing

4.2 Layer 2: Controller (Arduino)

Here the main aim was to design such an embedded controller which reduces vibration. Each motor is controlled by different Arduino via motor driver. Motor driver accepts pulse input to increment position of motor as well as direction signal to set direction of rotation. Overall architecture of controller is as shown in figure 6. Arduino Mega is used as central controller which receives command from ROS and then sends it to other arduinos. It is also responsible for collecting homing sensor data and sending it back to ROS. Each arduino receives number of pulses (based on angle to move) and direction. These commands are then processed and precise and continuous pulses are generated using internal Arduino TIMERS which are send to motor driver.

4.2.1 Command Set specification

Command set sent from ROS consists of total 11bytes of data as described.

(B1,B2) Number of pulses for motor 1

(B3,B4) Number of pulses for motor 2

(B5,B6) Number of pulses for motor 3

(B7,B8) Number of pulses for motor 4

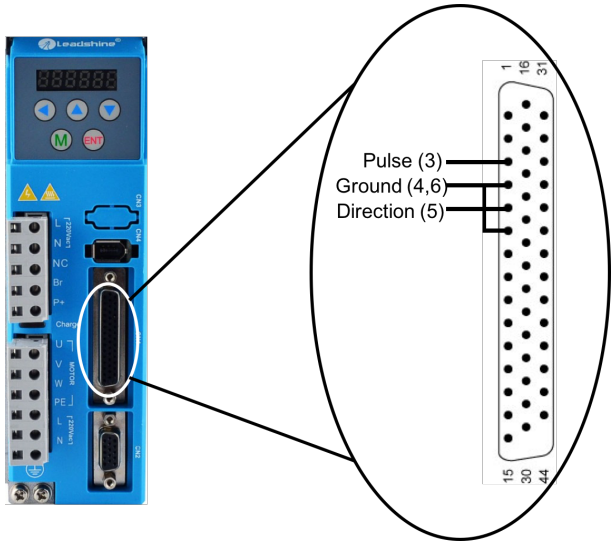


Figure 5: Control Signal connections in motor driver (44 pin D-sub connector)

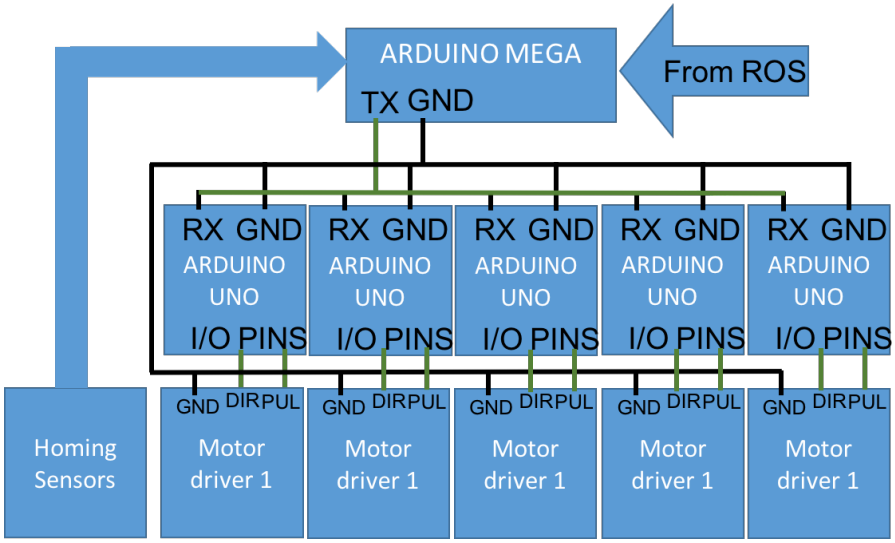


Figure 6: Controller Architecture and connection with robot

(B9,B10) Number of pulses for motor 5

B11 This byte is used as direction byte. 5 bits in this byte are assigned for direction of each motor.

This command set is received by arduino mega and is forwarded to each arduino. Arduino discards irrelevant bytes and process bytes significant for that particular arduino.

4.2.2 Homing Sensors

Robot has five homing sensors, one for each joint. These sensors has arc type shape. Sensor values are based on whether joint is in limit of arc or outside arc. These values are measured by arduino Mega digital input pins. After some manipulation (to keep consistency among sensor readings) this data is sent back to ROS. Homing

sensors works on 24V thus a simple voltage divider built using resistors to drop voltage level to 5V.

4.2.3 Pulse generation algorithm (in Arduino)

This section covers the most difficult and most important part of controller. As already explained before we need to generate pulses of precise timing in order to control motors. Streaming rate is 100 i.e. a command set is received by arduino every 10 ms. Pulse is generated by flipping digital pin whenever timer1 overflows. We set timer1 to measure half time period of pulse. There is a counter for number of pulses which is decreased after one pulse is generated and is updated when next command is arrived. Main point in this is that, the pulse generation is not stopped even if counter is reduced to 0 i.e. even if somehow all pulses are generated even before next command arrives arduino keeps generating pulses. This is done to keep continuous pulses to reduce vibration. Thus in case more or less pulses are generated than required they are accounted in next command set. We update this counter based on next command set as well as previous number of pulses that were remained or extra. Reading command set is done using serial interrupt to make it a parallel process. When command set is read properly *dataReaded* flag is set. Arduino keeps on generating pulses until ROS sending commands or sends zero as number of pulses. In case ROS stops sending command it is detected after 50ms, i.e. arduino waits for 50ms for command to arrive. In case no command is received a *STOP_MOTION* flag is set. This starts a stop motion routine which slowly reduces the speed of motor from current speed to complete halt and arduino goes in waiting mode. As soon as new command is detected whole process start again.

4.3 Layer 1: Software interface (ROS)

Layer 1 is software interface for controlling robot which is based on ROS. Following sections would delineate the architecture and algorithms.

4.3.1 ROS-SERVER

ROS is used to control Robot via USB. Main node is *tal_brabo_driver*, this node is driver node and is responsible for interfacing ROS and Arduino. ROS graph structure can be seen in figure 2. Images from USB camera are grabbed by *usb_cam_node* which are further processed by *object_tracker* node to approximate 3D coordinates of center of ball used as object to be tracked by robot end-effector. Then based on these coordinates and model of robot *velocity_controller* nodes compute joint velocities and then updates joint angles of robot and publish new joint angles at */joint_command* topic. Based on joint angles published by velocity controller *tal_brabo_driver* decides what commands required to be send to Arduino and send them at specific streaming rate to Arduino via USB. It is important to note that here velocity controller can be replaced by any other controller based on task. Controller need to publish joint angles on topic */joint_command* topic at any rate smaller than or equal to STREAMING RATE of the *talbrabo_driver* which is 100 in default settings.

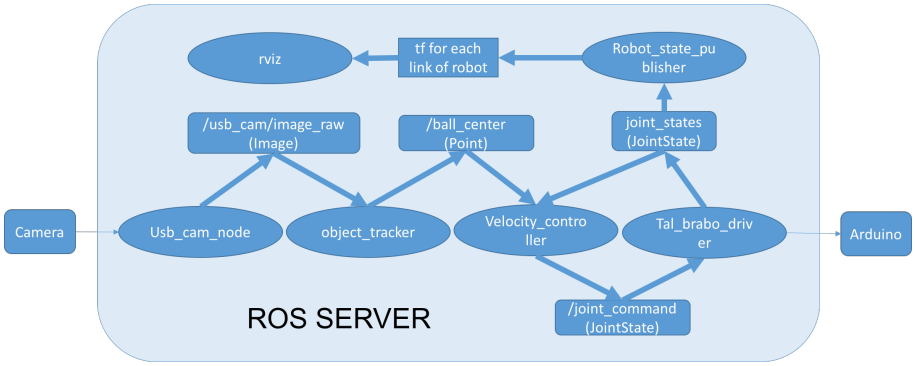


Figure 7: ROS server nodes graph

4.3.2 Position Streaming Interface

We are using Position Streaming interface for controlling robot. In this method, joint positions are streamed to the controller.Joint velocity is computed by the controller. ROS send command to controller at some fixed publishing rate (default is 100). This means we send command to controller every 10ms. Its job of controller to generate corresponding pulses and direction signal in a time frame of 10ms.

4.4 Node: tal_brabo_driver

This node is the main node responsible for sending commands to arduino based on joint angles received. This node is subscribed to */joint_command* topic having message type *sensor_msgs/JointState*. This node keeps track of joint angles, as there is no feedback available from robot about current state of robot. Thus user must note that restarting this node will first reset robot to its initial position. Following steps would describe the program (algorithm) used in it

- First serial connection is made with Arduino Mega via usb.
- Homing procedure starts
- User inputs are taken to get initial position of joints. Since homing sensor could only detect whether joint angle is on magnetic arc or not, these does not give exact position. Therefore user is asked to input joint location of first 3 joints, whether they are on black or white side of arc (Sides are marked on physical robot).
- Based upon user input joints are slowly rotated to achieve initial position (when joint are on one edge of arc). Joint 4 and joint 5 are rotated in full circle to reach edge of arc of homing senors.
- After homing robot is command to reach a different initial position required for task of object following.
- After this node starts waiting for messages on the topic */joint_command* to get target joint angles.

- As soon as it starts receiving commands it calculates pulses to be sent to arduino based on GEAR RATIO and angle to be moved. Also there is limit to maximum number of pulses to be sent to arduino for safety reasons.
- Directions of motors are calculated and finally a command set is transmitted. Also current joint angles are updated and published.
- Since motor4 and motor5 are connected to each other by a bevel gear, thus actual joint velocity of joint5 is dependent on joint4 velocity. Thus pulses are provided in such way to compensate for this. This compensation is based on following observation, Joint5 provided commanded is equal to joint5 desired velocity minus joint4 desired velocity.

4.5 node: velocity_controller

This node is responsible for publishing appropriate joint angles to *tal_brabo_driver* based on position of center of object tracked. This node is subscribed to *ball_center* to get update on center of object. This node is also subscribed to *joint_states* topic for getting feedback of current joint angles of robot. This node first calculates end-effector velocity of robot using simple PID algorithm.The error in PID is defined as difference between desired location (0,0,D) and current location of object. Desired location is set to (0,0,D) as it will ensure robot will try to move in such a way that object remains at center of camera and distance D away from it. Thus in case user tries to move object robot would try to follow the object.
These end effector velocities are then translated to joint velocities using inverse Jacobian. Inverse Jacobian is calculated based on current joint angles as received by this node. Target joint angles are calculated by adding these joint velocities to current join angles. Finally, these target joint angles are published on topic */joint_command*.

4.6 Node: object_tracker

This node detects green colored circular object in the image and calculates center of object. This node uses already built in method of blob detection in openCV. First image is thresholded based on HSV range of object. Then based on some threshold parameters best suited blob is selected and center is calculated. Node approximates Z distance as same as size of circular object. This approximation is valid only in case of circular objects. Since, velocity controller is based on PID therefore, there is no need to calculate distances in proper units which would require camera matrix and other things. Values of PID constants are tuned according to pixel level coordinates and size of object treated as distance in Z axis.

5 Results and Discussion

Robot was able to follow ball and whole system worked for prolonged time without any single system failure. Visualization on Rviz was in good sync with the real robot in all tasks. Following tasks were successfully completed by robot

Homing at given position hello

X-axis motion

Y-axis motion

Z-axis motion

Pure rotation of end effector

Ball follow in 2-d plane

Ball follow in 3-d

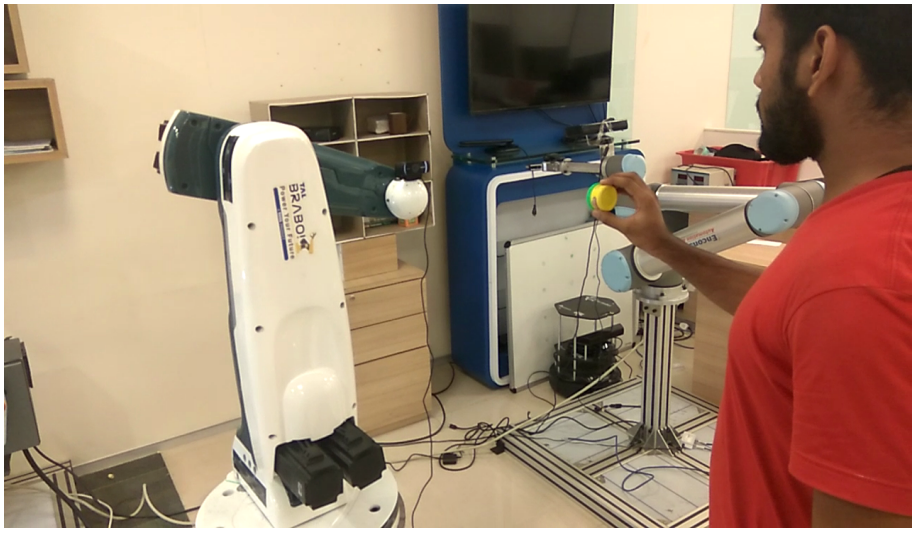


Figure 8: Robot following green colored cap

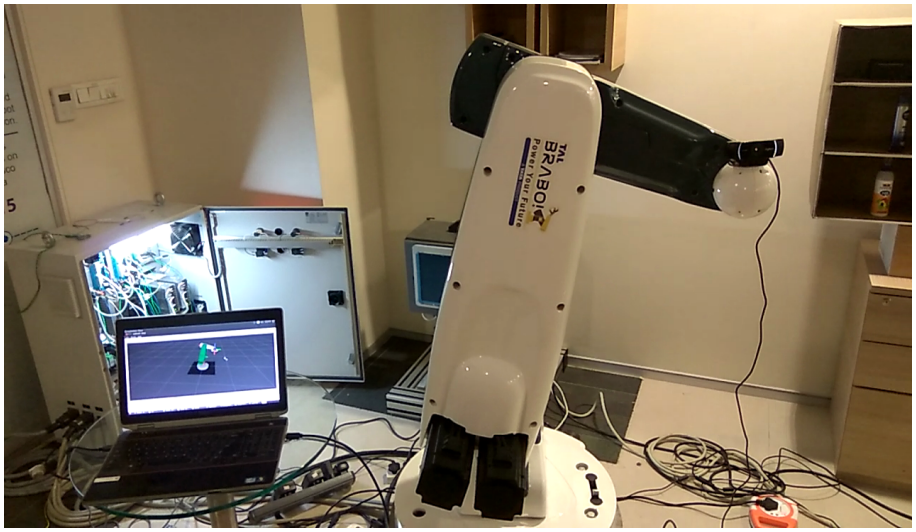


Figure 9: Visual feedback of current state of robot shown in rviz (laptop)

References

[1] Robot Operating System: https://en.wikipedia.org/wiki/Robot_Operating_System

[2] ROS Introduction: http://wiki.ros.org/ROS/Introduction

[3] ROS Concepts: http://wiki.ros.org/ROS/Concepts

[4] ROS Higher Level Concepts: http://wiki.ros.org/ROS/Higher-Level%20Concepts

[5] Arduino 101 Timers and Interrupts: http://robotshop.com/letsmakerobots/arduino-101-timers-and-interrupts