Android based teleoperation for the finch robot

FAUST, Oliver

Published version

FAUST, Oliver (2016). Android based teleoperation for the finch robot. ICTACT Journal on Communication Technology, 7 (3), 1334-1340.

ISSN: 2229-6948(ONLINE)
DOI: 10.21917/ijct.2016.0196

TUTORIAL ARTICLE

ANDROID BASED TELEOPERATION FOR THE FINCH ROBOT

Oliver Faust
Department of Engineering and Mathematics, Sheffield Hallam University, United Kingdom
E-mail: oliver.faust@gmail.com

Abstract

The act of creating a robot involves systems engineering and creative problem solutions. It is about using established components to create a system that works in the natural or at least in the human environment. The current project is no exception, we have used the Robot Operating System (ROS) to create an android based teleoperator application for the Finch robot. A Raspberry Pi processing platform establishes the link between the android device and the Finch robot. The most creative task, during the system design, was to translate the commands from the teleoperator application into wheel movements of the Finch robot. The translation must take into account the physical setup of the robot, including unintended negative influences, such as drag. The command translation involved a nonlinear coordinate transformation. The ROS framework enabled us to focus on that nonstandard coordinate translation task by offering a high level of abstraction and the ability to create component functionalities independently.

Keywords:
Robotics, Teleoperation, Robot Operating System, Coordinate Transform, Raspberry Pi

1. INTRODUCTION

Engineering is about building physical problem solutions [1, 2]. We have to use the tools and components which are currently available. The goal must be to limit the creation of both nonstandard functionality and nonstandard design processes. Nonstandard functionality is error prone and it gets outdated very fast. Furthermore, the documentation is local and oftentimes substandard. Non-standard design processes are error prone and most of them are not traceable in case something goes wrong with the design. Therefore, we have to rely on well thought out design methods, such as systems engineering [3] and establish the functionality with frameworks, such as Robot Operating System (ROS) [4] and the Application Programming Interface (API) of the Finch robot [5, 6].

Using the systems engineering design methodology [7, 8], in conjunction with ROS [4] and robot specific APIs, minimizes of the amount of creative energy needed to establish a physical problem solution. As a consequence, it is possible to focus the creative energy on designing interface solutions which rearrange and convert data. These interfaces have to be designed in a structured way. The design process starts with need definition, followed by requirements capturing and specification refinement. The implementation step translates the specification into a physical problem solution. The task of a designer is to come up with meaningful activities within each of these steps.

Building robots for education and research embodies the romantic idea of creation. It is incredibly satisfying to see that the will of the creator could animate otherwise lifeless material. In that paper, we describe the design steps which helped us to realize an android based teleoperation for the Finch robot. We applied systems engineering principles and design reuse through public and vendor specific frameworks. These design decisions allowed us to focus our creative energy on the design and implementation of a nonstandard functionality which interfaces ROS and Finch frameworks by translating messages. The nonstandard functionality took the form of a ROS node which communicates over rostopics and rosservices.

The material in the paper is arranged as follows. The materials section describes the system setup with both block and layer-diagrams. Subsequently, we focus on framing the problem for which we have to create a physical solution. From the problem analysis we distil the requirements. For that paper, the requirements take the form of a mathematical model. The discussion section provides some background on decisions which shaped the design process. The paper closes with conclusions and further work.

2. MATERIALS

As it is so often the case in the creative act of designing physical problem solutions, the creation of the android based teleoperation for the Finch robot started with a realization. To be specific, we realized that a Raspberry Pi [9, 10] can be used to power and control the Finch robot via the Universal Serial Bus (USB) interface. Furthermore, the Raspberry Pi can communicate via the internet through the WiFi interface. Hence, it immediately made sense to use the Raspberry Pi as a command and control gateway. That gateway links the user device to the Finch robot. We have chosen ROS, because it is the most widely used middleware for robots [11]. Subsequently, we have selected android based devices to establish the User Interface (UI). The reason for selecting that device class comes from the fact that only android mobile systems support the ROS middle-ware [12]. The Fig.1 shows the block diagram for the system, which facilitates the android based teleoperation functionality. The Raspberry Pi plays a central role in that setup, because it links the USB based Finch robot to the wireless android device. The icon on the left depicts an android device and the icon on the right depicts the Finch robot.

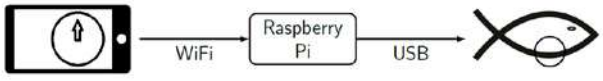

Fig.1. Block diagram for the Android based teleoperation system to control the Finch robot

A block diagram gives a broad overview of the system setup.

However, block diagrams are insufficient to establish the functionality, because the diagram hides too much complexity. The network layer diagram, shown in Fig.2, reveals the underlying functionality in terms of communicating components. All the components communicate in parallel, hence we had to establish a heterogeneous network, i.e. a processing and communication environment that stretches over multiple processing environments. For that particular project, the underlying processing environment was abstracted through the android system, the raspian Operating System (OS) and the Finch framework. On the positive side, most of that functionality is established through the ROS and the Finch framework [5]. As a consequence, the task for establishing the android based teleoperation functionality for the Finch robot comes down to interfacing the ROS with the Finch framework, in the Raspberry Pi.

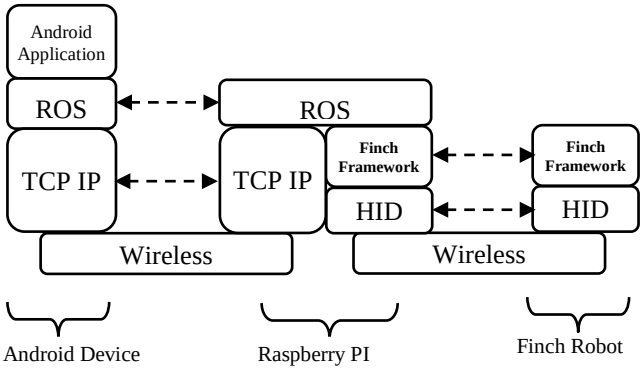

Fig.2. Layer model of the android based teleoperation system for the Finch robot

The ROS defines a node as a sequential entity that can make progress independently from other nodes. The nodes communicate via non-blocking rostopics or via blocking rosservices. Rostopics work on a many to many principle, whereas rosservices work on a one on one bases. Rostopics were used to exchange information between the android device and the Raspberry Pi. We have used rosservices to exchange information with the Finch framework within the Rasp-berry Pi. The following sections describe the mathematical foundation of the interface software which translates rostopic into rosservice messages.

2.1 TRANSLATING ROSTOPIC INTO ROSSERVICE MESSAGES

Before we can embark on establishing the transformation algorithms, we have to discuss the input and output data. The input data comes from the teleoperator app, executed by the ROS android framework in a mobile phone. The app provides a virtual joystick where the user can select a point within a unit circle. More specifically, the user touches the mobile device screen within a round area and these touch coordinates are translated into Cartesian coordinates within a unit circle. Whenever a user touches the screen region, symbolizing the unit circle, the app publishes rostopic messages. These messages are published periodically, i.e. there is a continuous message stream whenever a user touches the circular screen area. The messages have the format geometry msgs/Twist.msg [15]. Listing.1 shows an example message, as captured with the roscommand rostopic. Listing.1. Message from the teleoperation app, captured in the Raspberry Pi. The first message communicates the point $\left(\frac{\sqrt{2}}{2},\frac{\sqrt{2}}{2}\right)$ and the second message communicates (1,0).

```
i@boli~/roscatkin_ws$...
...rostopic echo/virtual_joystick/cmd_vel
linear:
x : 0.707106781185
y : 0.0
z : 0.0
angular :
x : 0.0
y : 0.0
z : 0.707106781185
---
linear:
x : 1
y : 0.0
z : 0.0
angular:
x : 0.0
y : 0.0
z : 0.0
---
```

The Finch is a two wheel plus support robot. The two front wheels can move independently forward and backward. The stand is at the back of the robot where it provides mechanical support. To move the robot, we need to communicate an integer value between -255 and +255 for each wheel via a rosservice. A value of 255 means maximum speed forward conversely a value of -255 means maximum speed backwards. The values between the positive and negative maximum are linearly related to the wheel speed. A value of 0 means the Finch wheel stops.

With that background, we embark upon the description of an algorithm which translates the geometry messages from the teleoperator app to wheel speed commands for the Finch robot. The diagram, shown in Fig.3, visualizes the relationship between the Cartesian coordinates, published by the teleoperator app, and the integer based wheel movement of the Finch robot. The diagram shows a unit circle centred at the origin of a two dimensional plane. There are eight points on that unit circle. The Cartesian coordinates are given in terms of (x,y). The second number pair, depicted as $(magnitude \angle phase)$, is the polar coordinate representation of that point. The numbers, that come from within the unit circle, describe the angle of the points in terms of [0, 2], where 0 is the same as 0 rad. The tuple, in the square brackets, represents the intensity of the Finch wheel speed in terms of [-1, 1]. The first number of the tuple describes the left wheel speed and the second number describes the right wheel speed.

The diagram, shown in Fig.3, relates the different representations of eight strategically chosen points on the unit circle. However, to design effective algorithms, such a graphical representation is too cluttered. A table brings out the rhythm of the point representation much better. Each row in Table.1 relates Cartesian coordinates, polar coordinates, normalized Angle and Finch wheel speed. The design pattern emerges from the last three columns. The normalized angle is conveniently mapped to the Finch wheel speed.

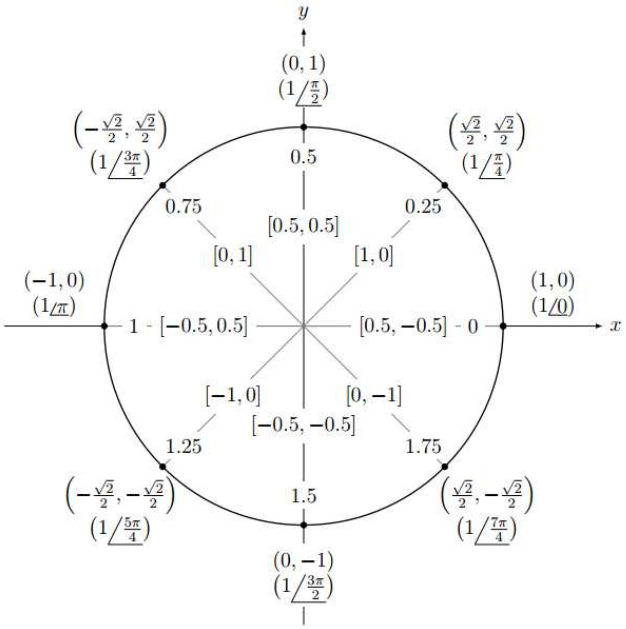

Fig.3. Transformation of rostopic messages from the android teleoperator app to Finch wheel speed commands. The diagram shows eight example points on the unit circle. Each point is labeled in terms of Cartesian (*x*, *y*) coordinates, (*magnitudes∠phase*), Angle and [Left Speed, Right Speed] for the Finch wheels.

Table.1. Coordinate transformation

Cartesian coordinates	Polar coordinates	Angle [0,2[	Speed	
			left	right
(1,0)	$(1\angle 0)$	0	0.5	-0.5
$\left(\frac{\sqrt{2}}{2},\frac{\sqrt{2}}{2}\right)$	$\left(1\angle\frac{\pi}{4}\right)$	0.25	1	0
(0,1)	$\left(1\angle\frac{\pi}{2}\right)$	0.5	0.5	0.5
$\left(-\frac{\sqrt{2}}{2},\frac{\sqrt{2}}{2}\right)$	$\left(1\angle\frac{3\pi}{4}\right)$	0.75	0	1
(-1,0)	$(1\angle\pi)$	1	-0.5	0.5
$\left(-\frac{\sqrt{2}}{2},-\frac{\sqrt{2}}{2}\right)$	$\left(1\angle\frac{5\pi}{4}\right)$	1.25	-1	0
(0,-1)	$\left(1\angle\frac{3\pi}{2}\right)$	1.5	-0.5	-0.5
$\left(\frac{\sqrt{2}}{2},-\frac{\sqrt{2}}{2}\right)$	$\left(1\angle\frac{7\pi}{4}\right)$	1.75	0	-1

Both, the diagram shown in Fig.3 and the information shown in Table.1, provide only a limited number of points. However, these points were chosen such that they represent important corner cases. Having established these corner cases allows us to interpolate the missing points for the wheel speed and thereby establish a functional relationship between the normalized Angle and the Finch wheel speed. The two graphs, shown in Fig.4, depict that functional relationship. To be specific, the graph for the left wheel is a triangular wave which oscillates between -1 and 1. The wave has a phase shift of 0.25 and a period length of 2. The triangular wave, which describes the right wheel speed, has the same period length and the same amplitude, but a phase shift of -0.25.

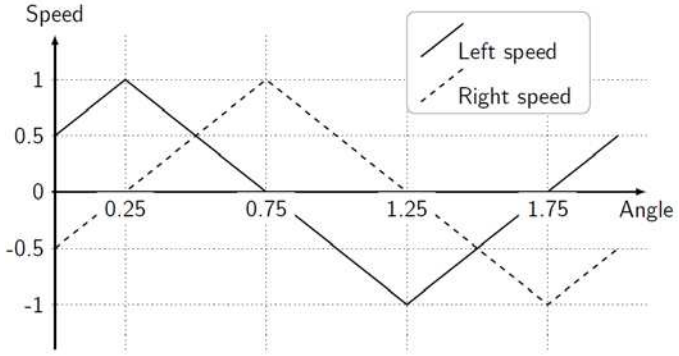

Fig.4. Transfer diagram from Angle to Left and Right speed.

Based on the transfer diagram, shown in Fig.4, we embark on the mathematical formulation of the transformation algorithm which converts Cartesian coordinates to Finch wheel speed. The first step in that mathematical model is to extract the angle from the Cartesian coordinates *x* and *y*. The Eq.(1) defines a function which yields the angle [-π..π] from *xy* coordinates.

$$\text{atan2}(x,y)=\begin{cases}\arctan\left(\frac{y}{x}\right) & \text{if } x>0,\\ \arctan\left(\frac{y}{x}\right)+\pi & \text{if } x<0 \text{ and } y\geq 0,\\ \arctan\left(\frac{y}{x}\right)-\pi & \text{if } x<0 \text{ and } y<0,\\ +\frac{\pi}{2} & \text{if } x=0 \text{ and } y>0,\\ -\frac{\pi}{2} & \text{if } x=0 \text{ and } y<0,\\ \text{undefined} & \text{if } x=0 \text{ and } y=0,\end{cases} \quad (1)$$

The resulting angle, in the range of [-π,π], is a bit awkward to deal with from an implementation viewpoint. To simplify the conceptional treatment of the translation process, we introduced the Angle [0,2] parameter. The Eq.(2) maps the atan2(*y*, *x*) results onto Angle.

$$Angle=\begin{cases}\text{atan2}(x,y)/\pi & \text{if } \text{atan2}(x,y)\leq\pi\\ 2-\text{atan2}(x,y)/\pi & \text{if } \text{atan2}(x,y)>\pi\end{cases} \quad (2)$$

The graph, shown in Fig.4, describes how the Angle value is translated into a Speed value. The Eq.(3) models that translation for the Left wheel speed.

$$Left\ speed = Speed\left(Angle\right) = \begin{cases} 2\times Angle & If\ \ Angle \le 0.25 \\ 2\times Angle + 1.5 & If\ \ 0.25 < Angle \le 1.25 \\ 2\times Angle - 3.5 & If\ \ Angle > 1.25 \end{cases} \quad (3)$$

Having a functional relationship between Angle and Left speed allows us to formulate the Right speed as a shift of the Left speed signal. The Eq.(4) shows that relationship.

$$Right\ speed = Speed\left(Angle - 0.5\right) \quad (4)$$

where, -0.5 is the phase shift value, which delays the wave and thereby shifting it towards positive infinity.

The Eq.(3) and Eq.(4) relate the Angle to both Left and Right wheel speed, as specified in Fig.3 and Table.1. However, these relationships hold only for points on the unit circle, i.e. for points where the distance to the origin is 1. In terms of robot speed that is the maximum condition. That maximum condition must be scaled down based on the actual distance of the point to the origin of the Cartesian coordinate system. The Eq.(5) defines r as the distance of the point (x,y) to the origin.

$$r = \sqrt{x^2 - y^2} \quad (5)$$

In our model, the Left and Right speed is scaled by r. In a final step, the scaled left and right speed is mapped onto the Finch wheel speed range.

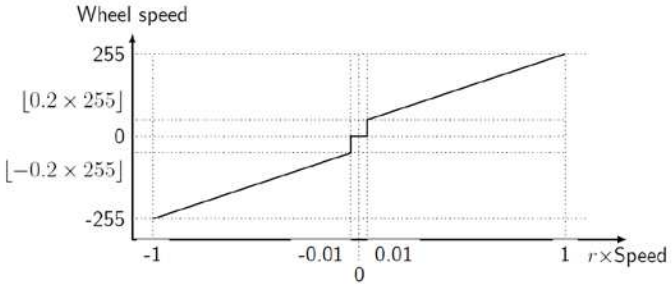

Fig.5. Transfer diagram from r×Speed to Wheel speed [-255, 254,..., 255].

The transfer function, shown in Fig.5, depicts the relationship between Finch wheel speed and r × speed. The values on the x-axis are continuous while the values on the y-axis are discrete steps. The transfer characteristic shows a discontinuity around the origin. That discontinuity was introduced because the Finch robot has only two wheels and one fixed plastic support at the back. The support has considerable drag and the driving motors need to generate sufficient torque to overcome that. Through experiments, we found that the value, when the Finch robot starts moving, is r = 0.2. Hence, whenever r × speed is greater ±0.01, then the wheel speed value jumps to $\lfloor \pm 0.2 \times 255 \rfloor$. That ensures a sensitive response of the robot to even very fine manipulations on the virtual joystick control.

$$Wheel\ speed = \begin{cases} \lfloor 255\times r\times Speed\times(1-0.2)+0.2 \rfloor & If\ \ \left|r\times Speed \ge 0.01\right| \\ 0 & If\ \ \left|r\times Speed < 0.01\right| \end{cases} \quad (6)$$

where, $\lfloor .. \rfloor$ indicate the floor function and 255 is the maximum value of the Finch wheel speed.

The Eq.(6) concludes the mathematical formulation of an algorithm model which translates massages from the android teleoperator app to Finch wheel speed. Translating the mathematical model to a software implementation is straight forward. Therefore, we have limited our explanations to the mathematical model rather than the implementation. Next, we discuss background information to shed some light on particular design decisions.

3. DISCUSSION

Robot design is to a large extend systems engineering with the goal of establishing a desired functionality with minimum development effort [13]. Therefore, frameworks are so important. To realize the android based teleoperation functionality for the Finch robot, we had to deal with two frameworks. The Finch robot comes with a C based API for the Raspberry Pi processing platform. That API was incorporated into the ROS framework. Accessing the Finch framework from ROS was facilitated through blocking rosservices. The user interface was established by making minor modifications to the ROSJAVA [14] example app called TeleOp. The app communicates via non-blocking rosservices. That approach allowed us to focus on the translation of the control messages from the android app to commands for the Finch robot.

As such, the requirements for the message translation were described in mathematical language. The purpose of the requirements capturing was not to achieve or establish mathematical rigor, it was to communicate the idea of how to translate the control messages to wheel movement. The mathematical equations, that were used to describe that process, were crafted with the source code specification in mind. In other words, the equations presented in the Materials section can be translated easily into implementation source code. For example, Eq.(1), Eq.(2), Eq.(3) and Eq.(6) contain cases which can be implemented with if ... then ... else statements. Furthermore, the definition of atan2(x,y), established with Eq.(1), is the same as the functionality specified in the math.h library for C and C++ [15].

Teleoperated systems have been used in a diverse range of applications, from biomedical, surveillance to manufacturing and service industries. Yepes et al. introduced an Android OS based application to control an industrial robot. To evaluate their system, a survey was done with engineering related users [16].

Bouterra et al. focused on the development of distributed architecture control for industrial robots. The developed design is a real time multi-tasking control law system. The control architecture was distributed in five microcontrollers with a master slave scheme. The master unit is dedicated for network management and communication with the user-interface. Each slave board focuses on the position control of the corresponding joint. The control was done through an USB communication assured with compatible drivers on the three most popular platforms (windows, Linux and Mac OS) [17].

Mauledoux et al. describe the use of software-in-the-loop and ROS as tools for controller implementation and simulation of discrete-time plants [18].

San-Millan presents a climbing robot with wheeled locomotion which uses permanent magnets as adhesion

mechanism. The robot is intended for the inspection of various types of ferromagnetic structures, such as ship hulls, wind turbine towers, bridges, and fuel tanks, in order to detect surface faults or cracks caused by, for example aging or atmospheric corrosion [19].

Teixeira et al. proposed the use of a wearable device for visualization and control in association with an Unmanned Aerial Vehicle (UAV) applied to the structural inspection of buildings [20].

In many assistive robotic systems, the interface to the user is simply a tablet computer or a monitor attached to a single robot. Benavidez et al. designed a software interface to connect users to an assistive robot system for the disabled and elderly. The system is comprised of heterogeneous low-cost assistive robots, a home management portal and a cloud computing backend [21].

4. CONCLUSION

This paper describes the creation of a nonstandard functionality to interface the ROS framework with the Finch API. To design that nonstandard functionality, we adopted the systems engineering methodology. In order to establish the desired functionality we had to translate ROS control commands to Finch wheel speed. The requirements were captured in the form of a mathematical model. The subsequent specification refinement, through source code and implementation as a Raspberry Pi program, was not explicitly described, because these steps are standard and as such the description would take up too much space in the paper.

Despite the fact that the material section just focuses on the mathematical model, used for requirements capturing, we were able to establish important points, which hold true for robot design in general. First and foremost, robot design is an exercise in systems engineering with the clear goal to establish a desired functionality with as little effort and as formal as possible. The mathematical requirements model turned out to be beautiful in its simplicity. Beauty made the content accessible and understandable. Having such a dependable and understandable formulation of the functionality made the implementation straight forward. The second major point is about focus and how to distribute the creative energy. While designing the system, we experienced that a clear methodology helped us to focus on the unique problem. The design success supports our belief that the focus, established through the design methodology, has helped us to overcome the difficulties. We were able to create a physical robot which functioned according to specification.

In future we will see higher and higher levels of integration. To reach these levels of integration there is an epic struggle going on in the background between closed and open source. The creation of the android based teleoperation functionality for the Finch robot involved both open and closed source components. Such an approach requires well documented APIs on both sides. We predict, that the coexistence between open- and closed-source will continue. That statement holds true especially for the field of robotics, where companies will continue to manufacture hardware with closed source software. The hardware acts as software verification, i.e. it is impossible to use the software without the hardware and vice versa. However, from the systems engineering perspective, such a scenario is not a problem, because there are only two questions for the selection of a component: Is the component affordable? Does the component meet the requirements? A particular component is selected based on the answer to these questions. As a consequence, framing robot design in terms of systems engineering is more important than to look for open- or closed-source.

Acronyms

API	Application Programming Interface
OS	Operating System
ROS	Robot Operating System
UAV	Unmanned Aerial Vehicle
UI	User Interface
USB	Universal Serial Bus

REFERENCES

[1] Oliver Faust, Ravindra Shetty, S. Vinitha Sree, Sripathi Acharya, U. Rajendra Acharya, E.Y.K. Ng, Chua Kok Poo and Jasjit Suri, "Towards the Systematic Development of Medical Networking Technology", *Journal of Medical Systems*, Vol. 35, No. 6, pp. 1431-1445, 2011.

[2] Zhe Song, Zhongkai Ji, Jian Guo Ma, Bernhard Sputh, U. Rajendra Acharya and Oliver Faust, "A Systematic Approach to Embedded Biomedical Decision Making", *Computer Methods and Programs in Biomedicine*, Vol. 108, No. 2, pp. 656-664, 2012.

[3] Erik Hollnagel and David D. Woods, "*Joint Cognitive Systems: Foundations of Cognitive Systems Engineering*", CRC Press, 2005.

[4] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully. Foote, Jeremy Leibs, Eric Berger, Rob Wheeler and Andrew. Ng, "ROS: an Open-Source Robot Operating System", *Proceedings of ICRA Workshop on Open Source Software*, Vol. 3, pp. 1-6, 2009.

[5] Tom Lauwers and Illah Nourbakhsh, "Designing the Finch: Creating A Robot Aligned to Computer Science Concepts", *Proceedings of Symposium on Educational Advances in Artificial Intelligence*, pp. 1902-1907, 2010.

[6] Taha Ben Brahim, Daniela Marghitu and John Weaver, "A Survey on Robotic Educational Platforms for K-12", *Proceedings of World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education*, Vol. 2012, pp. 41-48, 2012.

[7] U.R. Acharya, O. Faust, D.N. Ghista, S.V. Sree, A.P.C. Alvin, S. Chattopadhyay, T.C. Lim, E.Y.K. Ng and W. Yu, "A Systems Approach to Cardiac Health Diagnosis", *Journal of Medical Imaging and Health Informatics*, Vol. 3, No. 2, pp. 261-267, 2013.

[8] Oliver Faust, U. Rajendra Acharya and Toshiyo Tamura, "Formal Design Methods for Reliable Computer-Aided Diagnosis: A Review", *IEEE Reviews in Biomedical Engineering*, Vol. 5, pp. 15-28, 2012.

[9] Warren Gay, "*Mastering the Raspberry Pi*", 1st Edition, Apress, 2014.

[10] Eben Upton and Gareth Halfacree, "*Raspberry Pi User Guide*", 1st Edition, John Wiley and Sons, 2014.

[11] John Kerr and Kevin Nickels, "Robot Operating Systems: Bridging the Gap between Human and Robot", *Proceedings of 44th Symposium on Southeastern in: System Theory*, pp. 99-104, 2012.

[12] Rafael V. Aroca, Antonio Pericles B.S. de Oliveira and Luiz Marcos G. Goncalves, "Towards Smarter Robots with Smartphones", *Proceedings of 5th Workshop in Applied Robotics and Automation, Robocontrol*, pp. 1-6, 2012.

[13] Andrew Speers, Parisa Mojiri Forooshani, Michael Dicke and Michael Jenkin, "Lightweight Tablet Devices for Command and Control of ROS-Enabled Robots", *Proceedings of 16th International Conference on in Advanced Robotics*, pp. 1-6, 2013.

[14] D. Kohler and K. Conley, "Rosjava-An Implementation of ROS in Pure Java with Android Support", Available at: https://github.com/rosjava/rosjava_core, Accessed on 2011.

[15] Definition, Available at: http://docs.ros.org/api/geometry_msgs/html/msg/Twist.html, Accessed on 2015.