LAPPEENRANTA UNIVERSITY OF TECHNOLOGY
LUT School of Energy Systems
LUT Mechanical Engineering

Kirill Romanov

DEVELOPMENT OF AN OMNI BASED TRACTION SYSTEM FOR A TELEOPERATED MOBILE ROBOT UTILIZING ROS PLATFORM

Examiner(s): Professor Heikki Handroos
D. Sc. Tech. Eng. Hamid Roozbahani

ABSTRACT

Lappeenranta University of Technology
LUT School of Energy Systems
LUT Mechanical Engineering

Kirill Romanov

Development of an OMNI Based Traction System for a Teleoperated Mobile Robot Utilizing ROS Platform

Master's thesis

2018

72 pages, 37 figures, 4 tables and 10 appendices

Examiners: Professor Heikki Handroos
D. Sc. Tech. Eng. Hamid Roozbahani

Keywords: Tele-operated mobile robot, Mobile robot traction control, Mecanum wheels, ROS, Omni-directional movement, Motor drive programming, Wi-Fi communication.

The traction system is one of the key systems of the mobile robot which determines its ability to move in confined spaces when it is needed. The TIERA robot was developed in the Laboratory of Intelligent Machines at Lappeenranta University of Technology to perform various maintenance tasks in hazardous environments. The thesis project is focused on developing the traction system software of the mobile robot and providing the possibility of its remote control from the operator station. To accomplish this task, the communications system developed on the basis of Wi-Fi technology, Robot Operating System (ROS) middleware and Secure Shell (SSH) technology have been used.

The mathematical model of the traction system for the robot platform with four Mecanum wheels is introduced. The influence of previous researchers' developments on the decisions taken in this thesis, as well as the peculiarities of working with ROS and hardware installed previously are explained. Features of the software code development and implementation taking into account the hardware used in the TIERA traction system as well as the requirement for its remote control from the operator station are discussed.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to everyone who ever helped me in carrying out this work. First of all, I want to thank my supervisors Professor Heikki Handroos and Dr. Hamid Roozbahani for the golden opportunity to study at Lappeenranta University of Technology, in general, and work on the TIERA robot project, in particular. Besides, I would like to thank Juha Koivisto and Mikko Rikkonen, the other members of the laboratory staff, for their invaluable technical advice and help. I would also like to express my sincerest gratitude to Miguel Cordero Sanchez who guided me through many obstacles while carrying out the work.

Moreover, I would like to thank all the staff of Samara National Research University who provided me with invaluable support in organizing my study program at LUT: the director of the Institute of Engines and Power Plants Aleksandr Ermakov and the vice director Dmitry Uglanov, members of the Automatic systems of power plants department and its head Evgeny Shakhmatov and his deputy Aleksandr Igolkin as well as Victor Sverbilov, who is responsible for the Double Degree program at Samara University. And, of course, I would like to thank Georgy Makaryants, my supervisor at Samara University, who taught me a lot over the years of my research activities and Prokhor Almurzin, my teacher at Samara University, who taught me to always stick to guns.

Special thanks to Aleksandr Lukin, my colleague, who has given me tremendous support in solving important issues, my other good friends in Lappeenranta: Andrei Nekrasevits, Ertugrul Mayadağli, Alina Byzova and Antti Ahomäki and my dearest friends from Samara who kept my spirit despite the huge distance between us.

Finally, I would like to send my deepest thanks to the most important part of my life – my family who supported me anytime and anywhere.

Kirill Romanov

Kirill Romanov

Lappeenranta 14.05.2018

TABLE OF CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

L	X-axis distance from each Mecanum wheel to the center of the platform
l	Y-axis distance from each Mecanum wheel to the center of the platform
R	External radius of the Mecanum wheel [m]
v_{ir}	Tangential velocity of the free roller [m/s]
v_r	Resultant velocity vector of the robot platform [m/s]
v_{ix}	Velocity of the wheel in the direction of the X-axis [m/s]
v_x	Velocity of the robot in the direction of the X-axis [m/s]
v_{iy}	Velocity of the respective wheel in the direction of the Y-axis [m/s]
v_y	Velocity of the robot in the direction of the Y-axis [m/s]
x	Coordinate on X-axis [m]
y	Coordinate on Y-axis [m]
α	Offset angle of the roller [º]
β	Angle of movements [º]
γ	Angle between roller and disk axes [º]
ω	Rotational speed of the robot [rad/s]
ω_i	Rotational speed of the respective wheel [rad/s]
AGV	Automated guided vehicles
API	Application Programming Interface
CANopen	Controller Area Network open (protocol)
DLL	Dynamic Link Library
EC	Electronic commutation
EPOS	Easy Positioning System
HTTP	Hypertext Transfer Protocol
LabVIEW	Laboratory Virtual Instrument Engineering Workbench
LiDAR	Light Detection and Ranging
LUT	Lappeenranta University of Technology
MRO	Maintenance and repair operations
OPRoS	Open Platform for Robotic Services

ORoCoS	Open Robot Control Software
OS	Operating system
PC	Personal computer
RAM	Random Access Memory
ROS	Robot Operating System
ROW	Roller joint on wheel
SSH	Secure Shell
TCPROS	Transmission Control Protocol for ROS
UDPROS	User Datagram Protocol for ROS
URDF	Universal Robot Definition File
URI	Uniform Resource Identifier
WOP	Wheel on platform
XML-RPC	Extensible Markup Language Remote Procedure Call

1 INTRODUCTION

At the present days robots have become an integral part of our lives. More and more mobile robots are utilized in many areas of our life and traction system is an essential part of them. Each type of the traction system has its own advantages and disadvantages so it is selected based on the requirements to the robot for maneuverability, controllability, stability, efficiency, maintenance, et cetera. Based on the traction methods, the robot can be divided into the following groups:

- Wheeled robots;
- Legged robot;
- Whole body robots;
- Hybrid robots.

To date, a wheel is the most common locomotion mechanism for the movement of both mobile robots, in particular, and most of the land transport, in general, as it can provide good power efficiency with a relatively simple mechanical implementation. Compared to legs, wheels are in constant contact with ground which allows them to avoid impacts and they are easy and inexpensive to construct and maintain. For this reason, most of the industrial mobile platforms and robots are equipped with wheels.

As it is implied in the title the focus of the present work is the traction control system of the wheeled TIERA mobile assembly robot designed in the Laboratory of Intelligent Machines at Lappeenranta University of Technology (LUT). This section introduces research background of the thesis project, the robot and the research problems to overcome.

1.1 Research background

Nowadays, there exist a great number of wheeled mobile robots. As the analysis of papers on the topic related to the traction system of the wheeled robots shows, this issue has become more and more relevant during the last 15 years which can be observed in Figure 1.1.

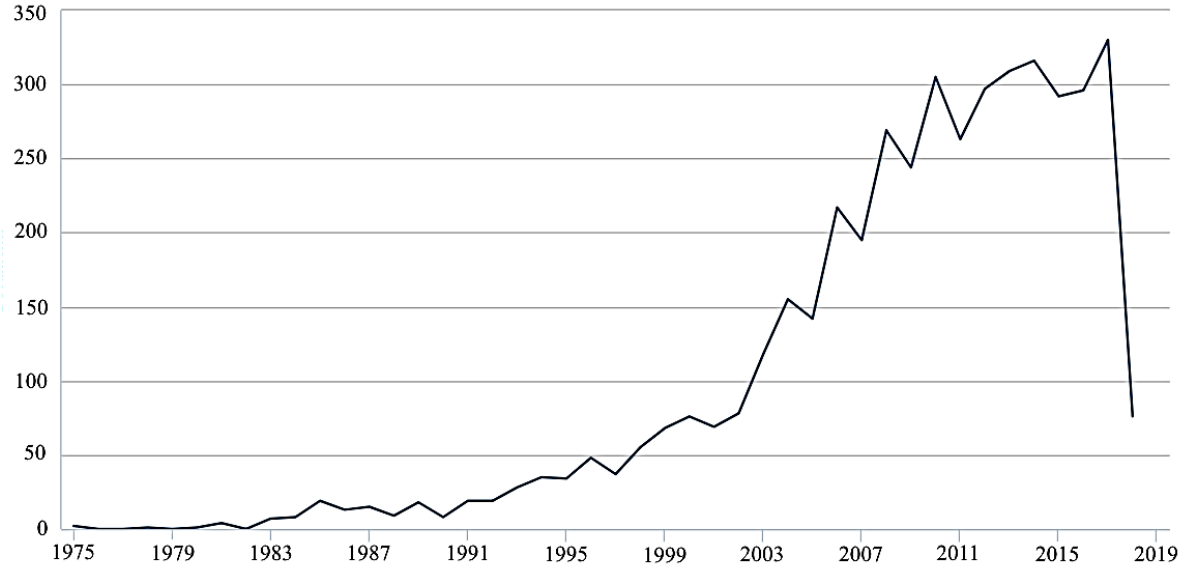


Figure 1.1. Papers per year by "robot wheel system" request (Scopus 2018a).

For instance, Cheong et al. (2016) designed a prototype of robot-waiters which are operated by a number of restaurants in Asia, Thale & de Villiers (2008) developed a mobile platform for materials handling and Gopalakrishnan et al. (2004) introduced the robot which is able to avoid obstacles and response to light and sonar inputs.

A large number of robots are utilized in diverse industries to perform various tasks, from simple to complex ones. Automated guided vehicles (AGV), which, according to Kalpakjiaan & Schmid (2012), have flexible functionality and can perform both the delivery of components and participate in the assembly operations, can be mentioned as an example of such robots.

The abovementioned examples demonstrate that good performance has been achieved in many applications of the mobile robots. Nevertheless, there still exist problems with the application of the mobile robots in industrial environments, such as chemical plants or nuclear stations, which become less human-friendly with the advancement of technology. Such robots require high accuracy and high maneuverability with the possibility of teleoperation at the same time.

In 2011 an earthquake near the Pacific coast of Japan and a following tsunami caused an industrial disaster at the Fukushima nuclear power plant. Its further development was prevented by 50 technicians at the expense of their health and lives (BBC 2013). One of the possible ways to avoid this situation in the future could be the remote use of mobile robots which would perform various maintenance and repair operations (MRO) in toxic, radioactive or other hazardous environments. The TIERA robot was designed as a prototype of such a robot.

At present time, most of the available industrial robots which are able to perform assembly and other mechanical operations are stationary and usually consist of a manipulator with a gripper fixed to a surface as those presented in the paper by Chen & Dong (2013). Thus, the development of the TIERA robot and, in particular, its traction system will solve a lot of pressing problems in industry and other areas of human activity such as MRO and vision in industrial disaster sites, bomb disposal during military operations and other operations performed in hazardous environments.

The primary goal of the thesis project is to develop a traction system for the TIERA robot which should be able to function in harsh environments and move in a restricted space. It is a prerequisite that the mobile robot is to be controlled remotely.

1.2 Description of the TIERA robot

The TIERA robot which is represented in Figure 1.2 can be subdivided into the following main parts:

- Embedded industrial computer;
- Wheels' traction system;
- Industrial arms and 3-finger adaptive robot grippers to carry out MRO;
- Head with an implemented vision system;
- Movable neck;
- Various sensors.

The structural arrangement and functionality of the robot are conditioned by the tasks it is to perform under influence of radiation, corrosion and threat of explosion, high voltages

and extreme temperatures. For example, a special industrial computer from Advantech, which is less sensitive to environmental conditions than conventional computers and will be discussed further, was built into the robot.

Figure 1.2. TIERA mobile robot.

The hardware of the TIERA robot is described in more detail by Belzunce (2015) and its simplified layout which reflects the relationship between the components of the robots can be observed in Figure 1.3. TIERA has been designed and developed at LUT by many generations of researchers. Accordingly, some technical decisions which influence the further research and development process were made by them. For instance, constructional design of the robot, in general, and the traction system, in particular, was developed by them. Furthermore, they decided to apply ROS middleware for communication and control of its systems and components and this must be taken into account when developing the control system for the robot wheels.

As TIERA is implied to be an omnidirectional mobile industrial robot, i.e. it should be able to move independently in longitudinal (forward/backward), lateral (right/left) and rotational directions, it was equipped with omnidirectional wheels which significantly improved its maneuverability. Typical omnidirectional wheels have been described by

many researchers and include Omni wheel (Song & Byun 2004), orthogonal wheel (Mourioux et al. 2006), tracked wheel (Ben-Tzvi et al. 2008) and Mecanum wheel (Muir and Neuman 1987).

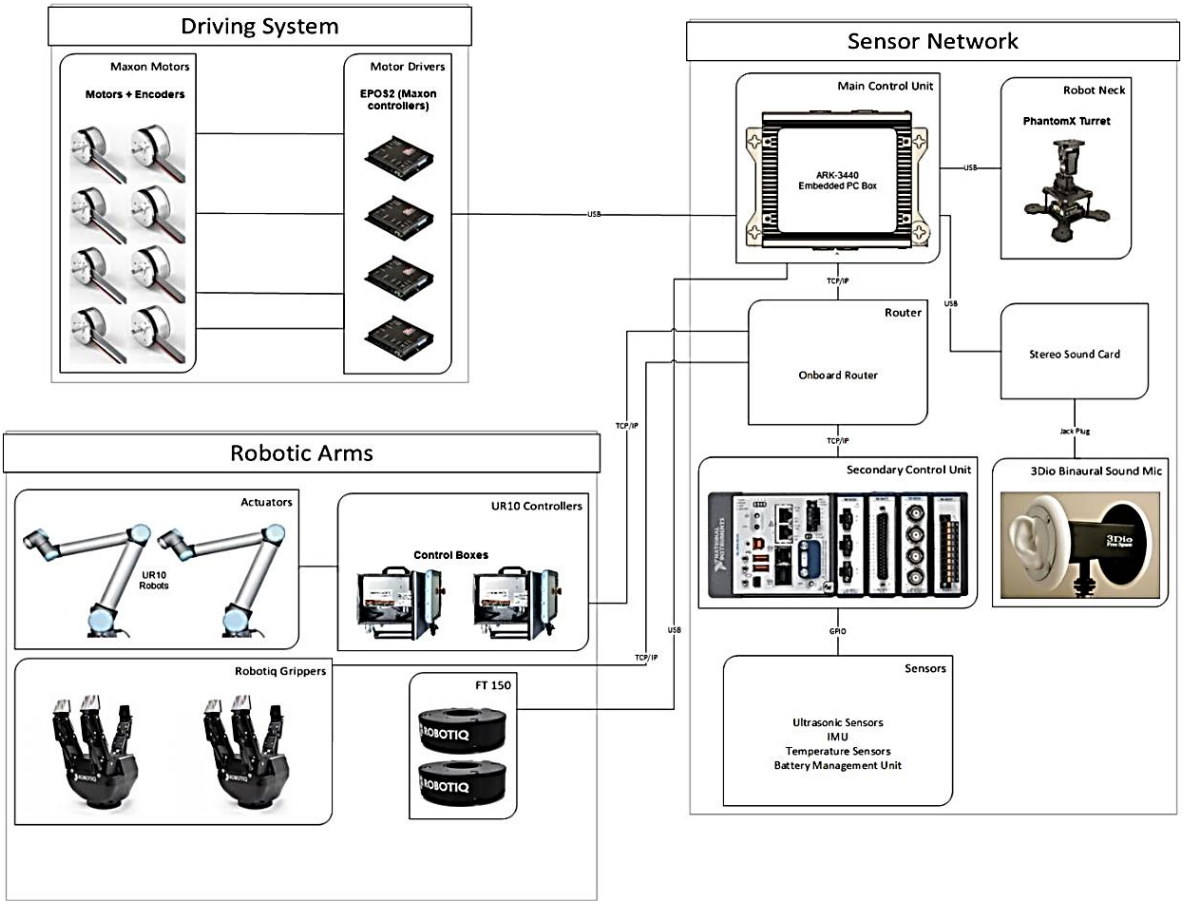


Figure 1.3. Layout of the TIERA hardware (Belzunce 2015).

Considering the performances of the above wheels and taking into account that kinematic and dynamic models of the Mecanum wheels have been widely developed by many researchers such as, Asama et al. (1995), Saha et al. (1995), Viboonchaicheep et al. (2003), Gfrerrer (2008), Lin & Shih (2013), et cetera, the previous LUT researchers decided to equip the mobile industrial robot with four 254mm (10 inch) aluminum Mecanum wheels. Their main advantage over the Omni wheels is they climb ramps easier and fit in a normal construction frame. An example of these wheels can be found in Figure 1.4.

The wheels are powered by Maxon motors which are equipped with magnetic safety brakes. Control of the wheels is performed using digital Easy Positioning System (EPOS) controllers. All components of the traction system are described further in more detail.

Figure 1.4. 254mm (10 inch) aluminum Mecanum wheel

1.3 Objectives to be achieved

Despite the fact that construction of the robot traction system was designed, it was still necessary to develop and implement the code of its control system as well as conduct ground tests. The developed traction system of the robot should provide the operator with the ability to control it from the specially equipped station and make the robot move in any direction in confined spaces.

As it was established during the project, the developed code required adjustments in order to set the remote control of the robot using communication systems developed by previous researcher Efim Poberezkin (2017) who was responsible for the communication system of TIERA.

The main objectives were set in the beginning of the work on the project and subsequently corrected taking into account the problems that arose in the course of the work. For

example, in order to avoid problems with overflow of Random Access Memory (RAM) in the digital controller it was decided to use RS-232 connection standard instead of USB for the reasons that will be described below. However, due to the differences in the robot control schemes between the computer, on which the primary code for the traction system was generated, and the embedded industrial computer, which directly controls the robot, the existing code was not able to establish the connection with the EPOS2 controllers via RS-232.

In addition, it was necessary to develop code that would allow one to quickly identify the problem and avoid these problems in the future, i.e. the debugging code. Moreover, the code developed by previous researchers did not release the safety brakes automatically after the initialization of the EPOS controllers. The ground tests were conducted upon completion of development and implementation of the traction system code.

1.4 Contribution of the thesis

One of the main results of the present Master's thesis is the development of the traction system for the TIERA mobile robot which can be controlled at a distance. Upon completion of the work, the robot should able to move in restricted spaces and conduct MRO under hazardous conditions. The operator should be able to control trajectory of the robot in space and monitor the velocity data calculated by the developed software.

The thesis also reflects possible ways of solving some specific problems (RS-232 connection standard, magnetic safety brakes, et cetera), which, for example, can be applied to the Maxon products since they are standardized and have similar characteristics and operating procedures and should prevent possible negative consequences.

2 METHODS AND EQUIPMENT

The present chapter will introduce the main methods and tools which have been used for the development of the traction system, such as the hardware which provides proper functioning of the system. Kinematics of traction system and ROS middleware, which is the main tool for programmatically connecting all the elements of the control system, will be described. Besides, the communication system developed by Efim Poberezkin (2017), which has been implemented into the TIERA traction system, will be also reflected.

2.1 Hardware of the traction system

Basic design calculations for the TIERA traction system were carried out by Le (2015) and do not represent much interest within the scope of this work. As it was briefly mentioned above, the traction system is formed by the following components:

- One industrial fan-less computer Advantech ARK-3440F-U5A2E;
- Four Maxon EC60 400W brushless motors with magnetic safety brakes;
- Four EPOS2 70/10 drive controllers;
- Four 254mm aluminum Mecanum wheels;
- Four GP 81 A planetary gears with 25:1 reduction rate;
- Eight flexible jaw-type shaft couplings;
- Four Power Gear P75L 1:1 L-transmissions;
- Four E-4BF-TRB 1 3/8 Timken tapered roller bearings;
- Four Controller Area Network open protocol (CANopen) cables for serial communication among the motors;
- One RS-232 cable for serial communication of the master drive with the Advantech computer;
- One Xbox controller for manual remote control from the operator station which is represented in Figure 2.1.

Each of the four wheel drives which control each wheel independently consists of the motors combined with planetary gears by default, which are followed by planetary gears with flexible coupling, L-transmissions with flexible coupling, the tapered roller bearing and the Mecanum wheels.

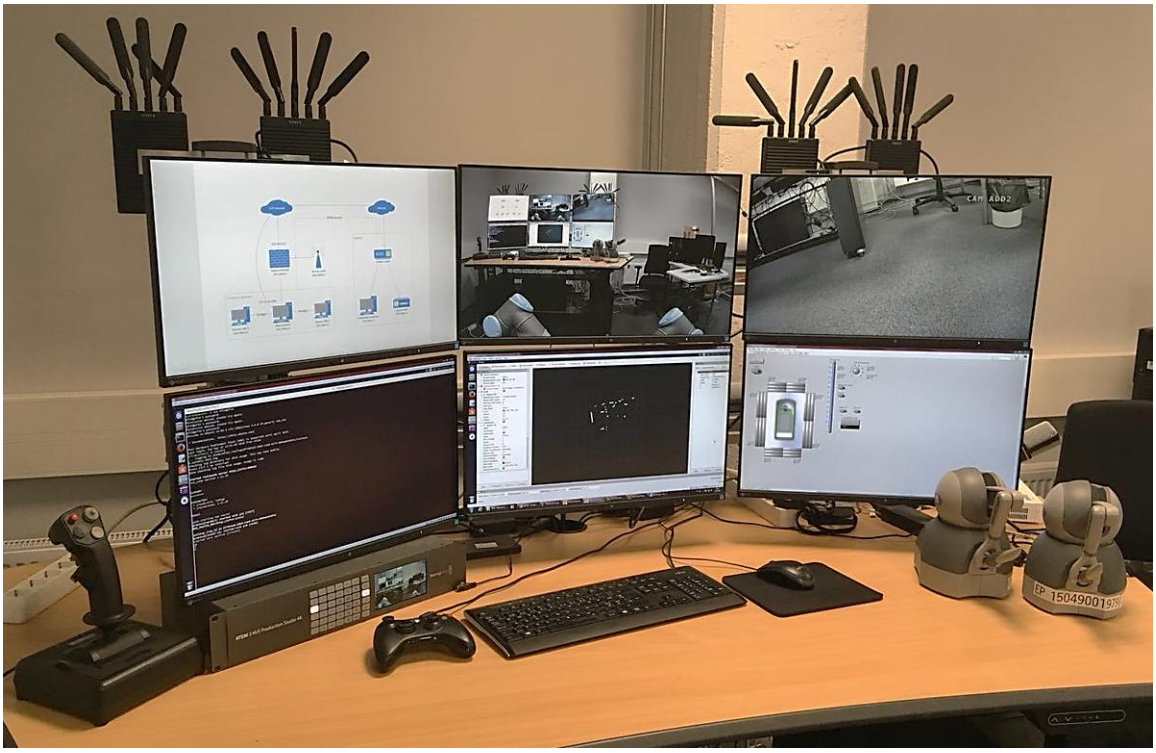


Figure 2.1. Operator station.

The complete system in the assembly can be seen in Figure 2.2. This arrangement provides individual control over each wheel, which allows independently adjust the speed of each wheel and the position of the robot in space. The main elements will be discussed below in more detail.

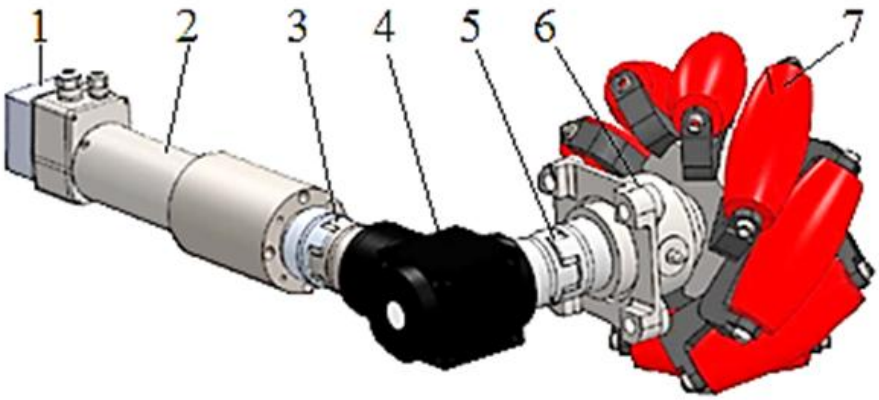


Figure 2.2. Wheel drive of the TIERA robot: 1 – motor, 2 – planetary gear, 3 – first shaft coupling, 4 – L-transmission, 5 – second shaft coupling, 6 – tapered roller bearing, 7 – Mecanum wheel (Le 2015).

2.1.1 Maxon motors

Each of the four 254mm aluminum Mecanum wheels is powered by its own Maxon electronic commutation (EC) motor. According to the official website (Maxon 2012), the main advantage of EC motors lies in their higher reliability, as well as higher speeds of rotation, since they are not limited by the mechanical commutation system. An example of the motor used in the TIERA traction system is depicted in Figure 2.3. Motor characteristics can be found on the official website of the supplier (Maxon 2018a). Some key characteristics of the motors are provided in appendix 1.

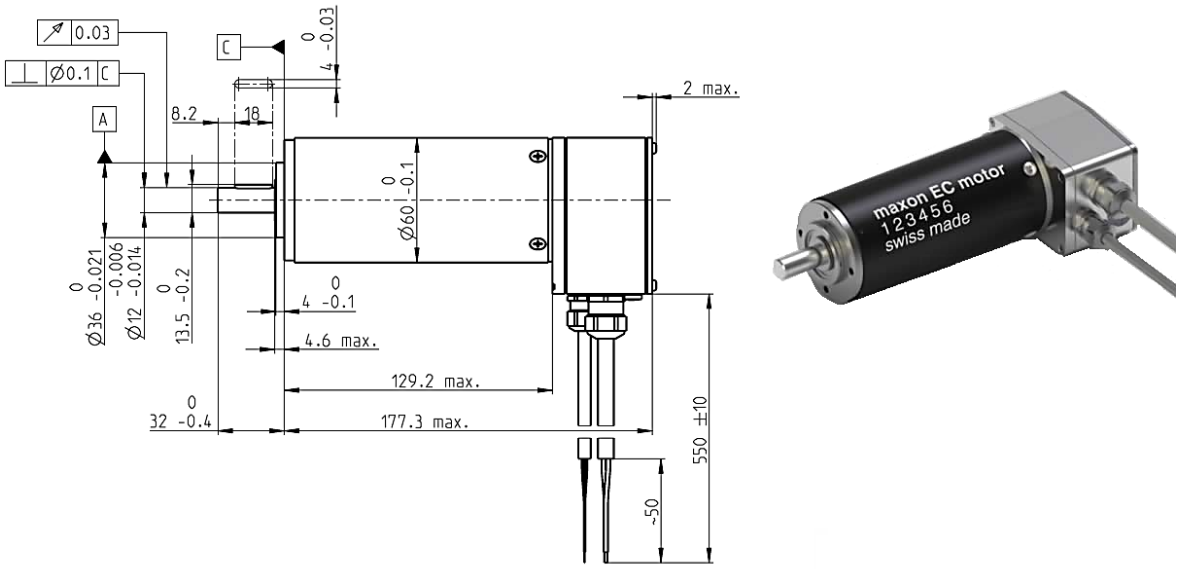


Figure 2.3. Schematic drawing (left) and external view (right) of Maxon EC60 400W brushless motor (Maxon 2018a).

2.1.2 EPOS2 controllers

Each motor is controlled by a separate EPOS2 70/10 digital positioning controller which is designed specifically them. There are several ways by which one can control the EPOS2 controllers, for example, utilizing Dynamic Link Libraries (DLL) or EPOS Studio software program in Windows or *epos_hardware* ROS package in Linux (Maxon 2018b). The controller, which is provided with 10 digital inputs, 5 digital outputs and 2 analog inputs, can be connected with DC and EC motors up to 700 W. Several drives can be combined via CANopen bus. (Maxon 2016d). Figure 2.4 depicts the wiring required for proper connection between the EPOS2 positioning controller and the EC motor with the encoder.

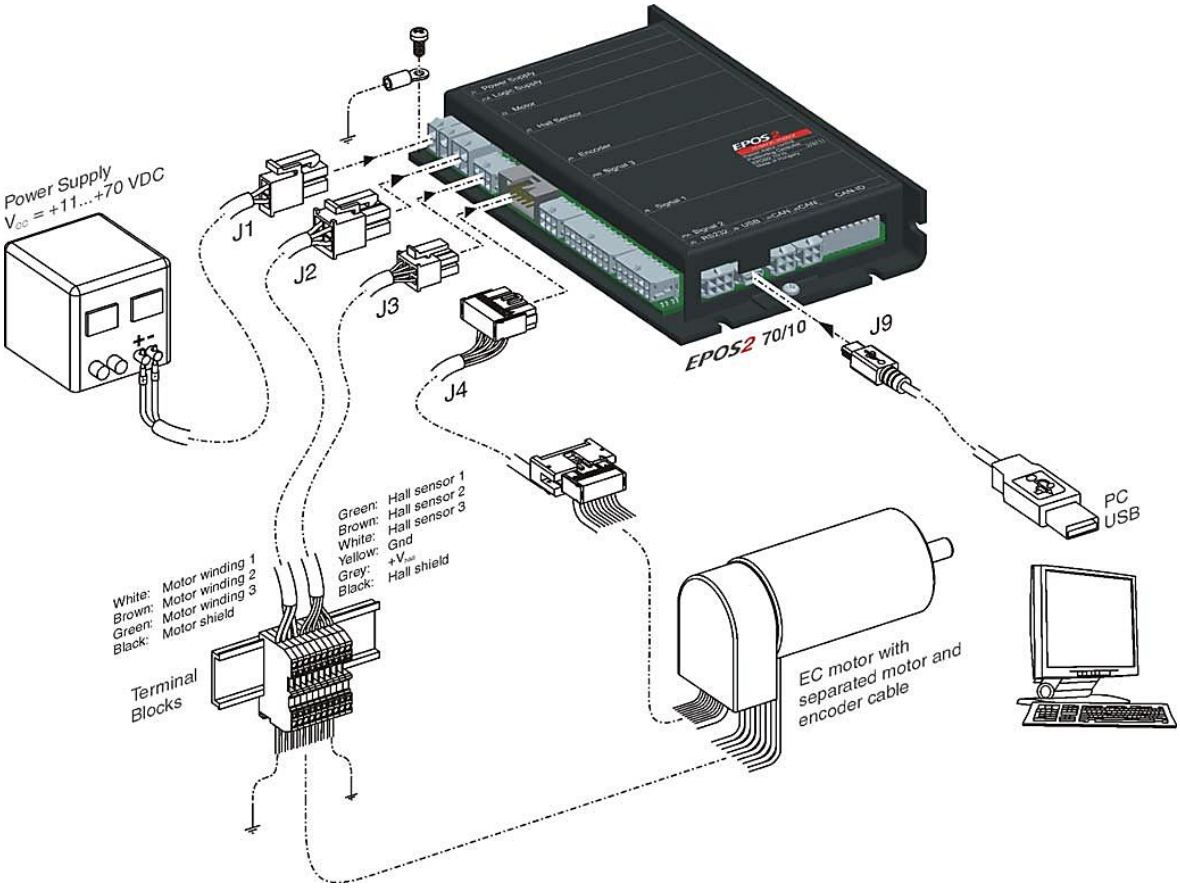


Figure 2.4. Minimum wiring for Maxon EC motor (Maxon 2016a).

As established in the official documentation, the EPOS2 70/10 controllers are developed to be commanded and controlled as slave nodes in a CANopen network or via any serial interfaces such as USB and RS-232 interface (Maxon 2016a). In the case of the TIERA robot, the network which is shown in Figure 2.5 consists of four EPOS2 70/10 controllers that control four motors.

The four controllers are connected in series using the CANopen protocol. The connection of the Advantech embedded computer with the first (master) controller, which controls the rear left wheel, is established using the RS-232 interface. The other three controllers are assumed to be slaves to the master controller. The blue dashed line indicates the CANopen connection between each motor and the grey line represents the RS-232 connection between the Advantech computer and the master motor drive.

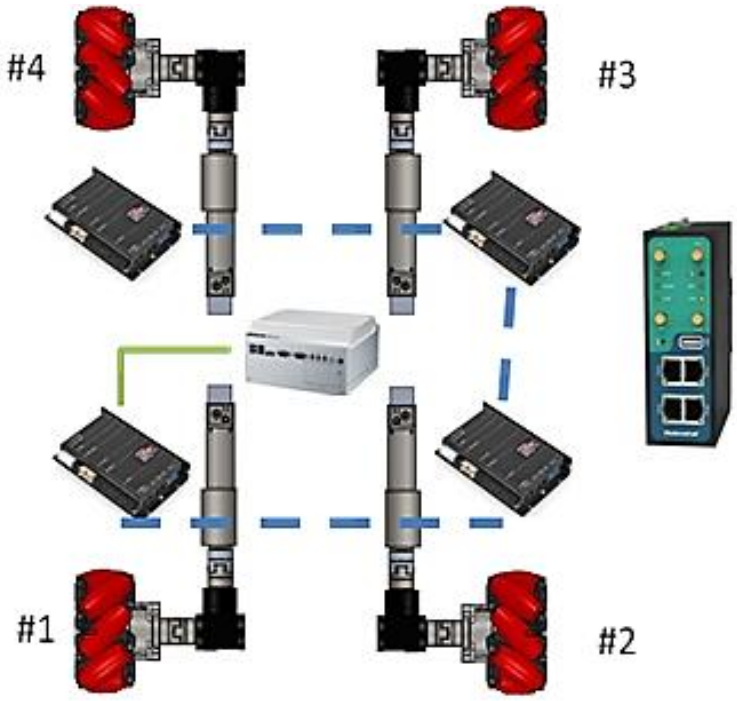


Figure 2.5. Traction system network.

2.1.3 Advantech industrial computer

The main control device of the whole robot is the Advantech ARK-3440F-U5A2E industrial computer which is shown in Figure 2.6. The main advantage of this computer is its high reliability and stability when working under severe conditions.

Key features of the computer which are presented on the official website of its distributor Walker Industrial are as follows (2018):

- Intel® Core™ i7 Fanless Embedded Box personal computer (PC).
- Multi-display and support for wide screen with high resolution.
- Supports 2 Giga bite Ethernet, eSATA, 6 USB 2.0, audio and 3 COM ports.
- Two internal 2.5" SATA HDD drive bays.
- Various expansion interfaces for diverse applications.
- Easy integration, easy maintenance, and wide input voltage range 9 ~ 34 V.
- IP40.
- Supports embedded software Application Programming Interfaces (API) and Utilities.

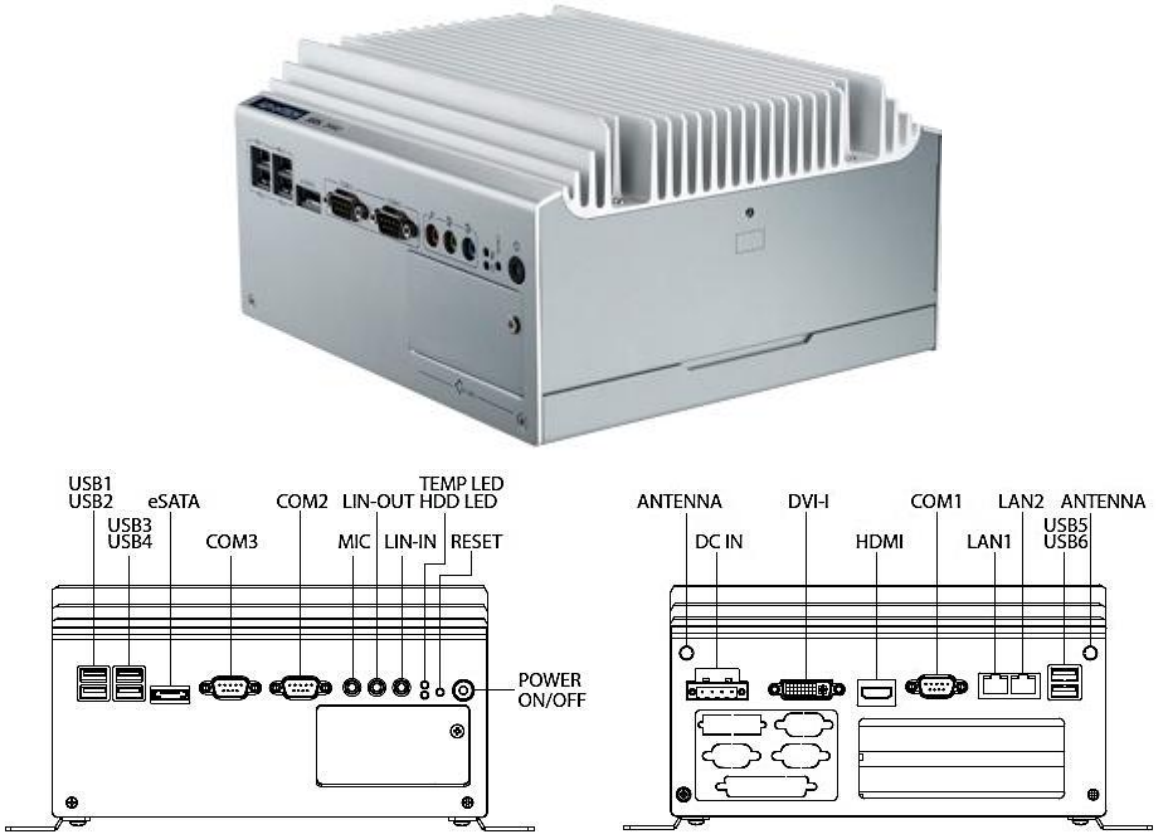


Figure 2.6. External view (top) and front (bottom left) and rear (bottom right) panel layouts of Advantech ARK-3440F-U5A2E fan-less embedded computer (Walker Industrial 2018).

Some basic technical characteristics of Advantech ARK-3440F-U5A2E can be traced from appendix 2. The rest specifications of the Advantech can also be found from the previous link. The computer supports both Windows and Linux OS and as it was said above Linux OS Ubuntu 14.04 is utilized for this project.

2.2 Kinematics of traction system

Mecanum wheel has become widespread in robotics as it provides good maneuverability in confined space conditions and high positioning accuracy if the kinematics model is correct. The problem of inverse kinematics is extremely relevant for the adequate and correct operation of the traveling system since the developer has to understand the principles of robot movement to create efficient control software.

The solution of the inverse kinematics problem allows one to calculate what actions the four wheels of the robot need to perform, so that it reaches the given position at the required angle. According to Le (2015), the maximum design load that the robot is able to withstand is 250 kg with the maximum velocity equal to 1 m/s and acceleration of 0.5 m/s^2.

2.2.1 Mecanum wheel

A standard Omni wheel, which, in addition to functioning as a conventional wheel, provides lower friction when moving in other directions, includes a disk with several conventional freely moving rollers placed on its periphery. The rollers are located at an angle of 45 degrees to the circumference of the disk in the Mecanum wheel, thus, shaping the silhouette of the wheel as circular. The roller has the shape of an ellipse which can be obtained by cutting the cylinder with the diameter of the wheel at angle γ between roller and disk axes equal to 45º and which geometry corresponds to the following formula:

$$\frac{1}{2}x^2 + y^2 - R^2 = 0 \tag{1}$$

where R is the external radius of the Mecanum wheel, x is the coordinate on X-axis and y is the coordinate on Y-axis. (Doroftei et al., 2008). The process of obtaining the angle γ can be seen in Figure 2.7.

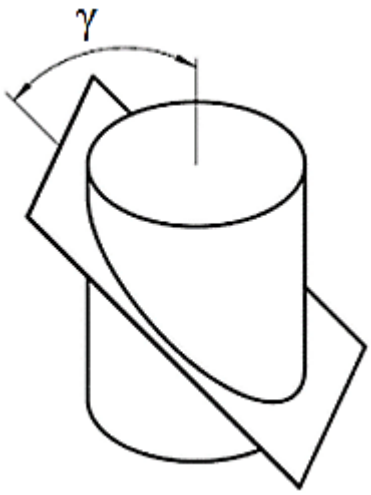


Figure 2.7. Process of obtaining the angle γ (Doroftei et al., 2008).

Further structural calculations of the wheel can be found in the Master's thesis by Weiting Le (2015) as they do not lie within the scope of the present project. The main features

worth noting that the rollers themselves are not actuated or sensed directly but the through appropriate motions of the motors. However, they increase the range of possible trajectories of the wheel.

The coordinate system rules followed while developing kinematics model are the right-hand approach and RGB rule (red axis – X, blue axis – Y and green axis – Z). The positive turn it is considered to be anti-clockwise. Figure 2.8 depicts the arrangement of the axes in the coordinate system of the contact roller.

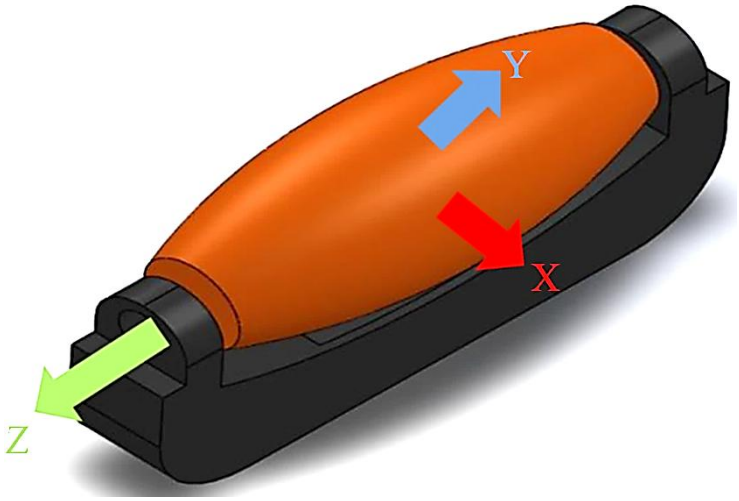


Figure 2.8. Coordinate axes of the contact roller.

The contact roller coordinate frame is associated to the roller which is “permanently in contact”, i.e. that there is just one assumed contact point that is on the vertical axis of the wheel. This assumption simplifies the logics behind the development of the kinematics equations.

All four Mecanum wheels are positioned symmetrically with respect to the center of the robot's platform for an even load distribution. Schematic representation of the wheel from which the equations of kinematics are derived can be seen in Figure 2.9.

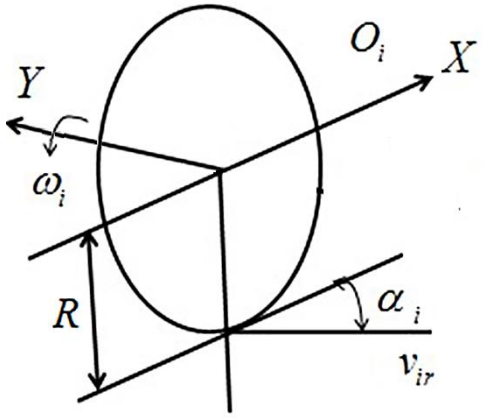


Figure 2.9. Schematic layout of the wheel (Jia et al. 2013).

Since each wheel is controlled by the separate motor, the robot is steered by combining the respective rotation speeds and, thus, moving the platform in the prescribed direction. The following velocity matrices can be derived from Figure 2.9 for wheels 1…4:

$$\begin{bmatrix} v_{ix} \\ v_{iy} \end{bmatrix} = \begin{bmatrix} R & \cos\alpha_i \\ 0 & \pm\sin\alpha_i \end{bmatrix} \begin{bmatrix} \omega_i \\ v_{ir} \end{bmatrix} \tag{2}$$

where i = 1...4, α is offset angle of the roller which is equal to γ for wheels 1 and 4 and -γ – for wheels 2 and 3, v_{ix} is velocity of the respective wheel in the direction of the X-axis, v_{iy} is velocity of the respective wheel in the direction of the Y-axis, ω_i is rotational speed of the wheel and v_{ir} is tangential velocity of the free roller which is in constant contact with the ground.

Analyzing the obtained matrices we can derive the velocity components for each wheel:

$$v_{ix} = \omega_i R + v_{ir}\cos\alpha_i \tag{3}$$

$$v_{iy} = \pm v_{ir}\sin\alpha_i \tag{4}$$

2.2.2 Robot platform

In order to calculate the kinematics of the robot it is 5 rigid-body coordinates systems should be taken into account:

- Contact roller;
- Roller joint on wheel (ROW);
- Wheel;
- Wheel on platform (WOP);
- Platform.

The platform is the coordinate frame associated with the robot platform. It is located at the middle position of the platform with estimated length of 1180 mm and width of 760 mm. The WOP coordinate system corresponds to the wheel location with fixed orientation with respect the platform. The wheel coordinate frame is located in the center of the wheel and rotates freely. The ROW coordinate axes are fixed in space within the wheel coordinate frame and have fixed orientation as Y-axis points to the center of the wheel. It is also assumed that there is one contact point on the vertical axis of the wheel. Considering all the above constraints, the kinematics equations of the robot can be simplified. Table 1 demonstrates the location of all the coordinate systems with respect to the platform. It is necessary for understanding how the transition from the coordinate systems of each individual wheel to the coordinate system of the entire robotic platform occurs.

Table 1. Location of the coordinate systems with respect to the platform.

Coordinate system	**x, mm**	**y, mm**	**z, mm**	**Roll, rad**	**Pitch, rad**	**Yaw, rad**
Platform	0	0	0	0	0	0
WOP 1	-590	315	0	0	0	0
WOP 2	-590	-315	0	0	0	0
WOP 3	590	-315	0	0	0	0
WOP 4	590	315	0	0	0	0
Wheel 1	-590	315	0	0	0	0
Wheel 2	-590	-315	0	0	0	π
Wheel 3	590	-315	0	0	0	π
Wheel 4	590	315	0	0	0	0
ROW 1	-590	315	-107.9	0	0	-π/4
ROW 2	-590	-315	-107.9	0	0	-3π/4
ROW 3	590	-315	-107.9	0	0	3π/4
ROW 4	590	315	-107.9	0	0	π/4

Layout of the robot necessary for further development of kinematics equations can be found in Figure 2.10 where L is X-axis distance from each Mecanum wheel to the center of the platform and l is Y-axis distance from each Mecanum wheel to the center of the platform. With reference to the kinematics of the entire robot, the following equations can be derived:

$$v_{ix} = v_x \pm l\omega \tag{5}$$

$$v_{iy} = v_y \pm L\omega \tag{6}$$

where v_x is velocity of the robot in the direction of the X-axis, v_y is velocity of the robot in the direction of the Y-axis and ω is rotational speed of the robot.

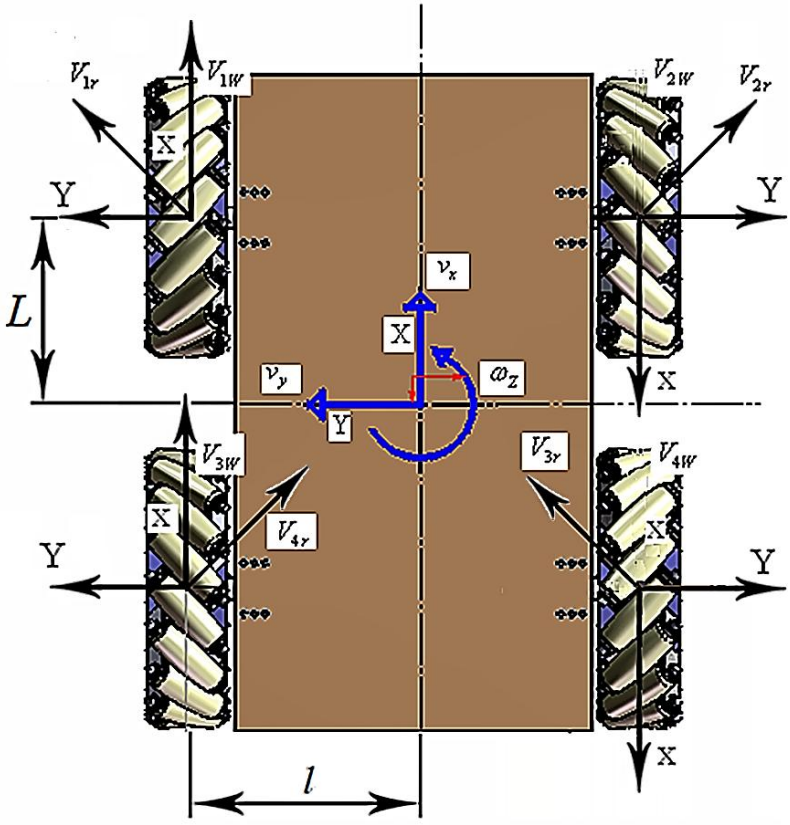


Figure 2.10. Layout of the robot platform (Wakchaure et al. 2011).

Considering that the robot moves on flat ground and combining (6) and (7), we can obtain the following matrix:

$$\begin{bmatrix} v_{ix} \\ v_{iy} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \pm L \\ 0 & 1 & \pm l \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} \tag{7}$$

By combining (2) and (7) the following matrix is obtained:

$$\begin{bmatrix} \omega_i \\ v_{ir} \end{bmatrix} = \begin{bmatrix} R & \cos\alpha_i \\ 0 & -\sin\alpha_i \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 & \pm L \\ 0 & 1 & \pm l \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} \tag{8}$$

Thus, the kinematic formulas of the corresponding wheels have the following form:

$$\omega_1 = \frac{1}{R}\left(v_x + v_y - (L+l)\omega\right) \tag{9}$$

$$\omega_2 = \frac{1}{R}\left(-v_x + v_y - (L+l)\omega\right) \tag{10}$$

$$\omega_3 = \frac{1}{R}\left(-v_x - v_y - (L+l)\omega\right) \tag{11}$$

$$\omega_4 = \frac{1}{R}\left(v_x - v_y - (L+l)\omega\right) \tag{12}$$

Taking into account the rotational speeds ω_i of the wheels are the only independent variable and $\alpha = 45^\circ$, the inverse kinematics matrix, from where the rotational speed of four Mecanum wheels are calculated, is as follows:

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} = \frac{1}{R} \begin{bmatrix} 1 & 1 & -(L+l) \\ -1 & 1 & -(L+l) \\ -1 & -1 & -(L+l) \\ 1 & -1 & -(L+l) \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} \tag{13}$$

This equation is fundamental for the development of the traction control system which will be described in more detail further. The direct kinematics matrix can be derived from (13):

$$\begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} = \frac{1}{4}\begin{bmatrix} 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ -\frac{1}{(L+l)} & -\frac{1}{(L+l)} & -\frac{1}{(L+l)} & -\frac{1}{(L+l)} \end{bmatrix}\begin{bmatrix} R\omega_1 \\ R\omega_2 \\ R\omega_3 \\ R\omega_4 \end{bmatrix} \tag{14}$$

Therefore, the velocity components of the robot platform are as follows:

$$v_x = \frac{R}{4}(\omega_1 - \omega_2 - \omega_3 + \omega_4) \tag{15}$$

$$v_y = \frac{R}{4}(\omega_1 + \omega_2 - \omega_3 - \omega_4) \tag{16}$$

$$\omega = \frac{R}{4(L+l)}(-\omega_1 - \omega_2 - \omega_3 - \omega_4) \tag{17}$$

From here one can calculate the module and the direction of the resultant velocity vector:

$$v_r = \sqrt{(v_x^2 + v_y^2)} \tag{18}$$

$$\beta = \tan^{-1}\left(\frac{v_x}{v_y}\right) \tag{19}$$

where β denotes the angle of movements and v_r denotes the resultant velocity vector of the robot platform. Based on these equations, one can draw the following conclusions:

- If $v_x \neq 0$, $v_y = 0$ and $\omega = 0$, the robot platform moves along X-axis. In order to achieve this trajectory the rotational speeds of wheels 1 and 4 and 2 and 3 should be v_x/R and $-v_x/R$, respectively.

- If $v_x = 0$, $v_y \neq 0$ and $\omega = 0$, the robot platform moves along Y-axis. In order to achieve this trajectory the rotational speeds of wheels 1 and 4 and 2 and 3 should be v_y/R and $-v_y/R$, respectively.

- If $v_x = 0$, $v_y = 0$ and $\omega \neq 0$, the robot platform rotates around the origin of the platform coordinate system. In order to achieve this trajectory the rotational speeds of wheels 1 and 4 and 2 and 3 should be $(L+l)\omega/R$ and $-(L+l)\omega/R$, respectively.

- If $v_x = v_r \cos\varphi$, $v_y = v_r \sin\varphi$ and $\omega = 0$, the robot platform moves without rotation along a line laid out by angle φ from the Y-axis. In order to achieve this trajectory the rotational speeds of wheels 1 and 4 and 2 and 3 should be $v_r(\cos\varphi + \sin\varphi/\tan\alpha)/R$ and $v_r(\cos\varphi - \sin\varphi/\tan\alpha)/R$, respectively.

2.3 Robot Operating System

In order for the robot to operate in aggressive environments, it is necessary to provide it with a big number of tools, embedded devices and other apparatus. As established by the Belzunce (2015) one of the optimal options for combining these features is implementing ROS which provides both the communication protocol and its implementation for the robot. Though there are other platforms like Robotics Technology Middleware (RT-middleware), Open Platform for Robotic Services (OPRoS) or Open Robot Control Software (ORoCoS), comparison of which can be observed, for example, in the paper by Magyar et al. (2015), the previous generation of researchers decided to use ROS because it can be easily utilized in complex robot projects.

According to official ROS website (2017): “ROS provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device

drivers, libraries, visualizers, message-passing, package management, and more." ROS also allows one to run parts of the code at different stations and, thereby, tele-operate the robot (ROS 2014a).

As ROS is an open-source platform for developing robot software most packages are developed by the users to control specific robot hardware and perform certain functions. Of particular interest is the *epos_hardware* package developed by Mitchell Wills that is responsible for initialization and control processes of the controllers and motors.

In addition, important is the fact that the ROS allows one to combine the code created in different languages including Python, C++ and Lisp in a single integral network (ROS 2014a). It should be also stressed that ROS is utilized exclusively on Linux operating systems (OS). For the above reason, Ubuntu 14.04 which supports ROS Indigo distribution (ROS 2016a) was installed on the Advantech industrial computer and the main station from which an operator controls the mobile robot. As mentioned by Poberezkin (2017), Linux was installed over Windows 7 using VirtualBox virtual machine on the main control station in order to avoid the IP conflict of the two grippers.

As stated on the ROS website (2014b), its concept consists of the File-system level, the Computation Graph level, and the Community level which are fundamental elements when developing code for different purposes in ROS. The Community level covers the possible resources provided by ROS community through which different groups of developers can exchange experiences, solutions to various issues and software. The other two levels will be discussed in more detail hereinafter.

2.3.1 ROS File-system Level

This level covers the types of files with which the user directly interacts on the stations, i.e. the ROS resources (ROS 2014b) which are listed as follows:

- Packages.
- Meta-packages.
- Package Manifests.

- Message types.
- Service types.

The main structural element of the code is a package which can consist of nodes, libraries supported by ROS, datasets or configuration files. The ROS code is organized using modular approach which implies that each package performs a specific function (for example, motor control, joystick wrapper, et cetera) in ROS workspace.

Meta-packages consist of packages that are combined together to perform certain operations with individual elements of the robot hardware and also to simplify code perception. Manifests, which are always represented by the *package.xml* file, provide metadata about a package, including its name, version, description, et cetera, and declares the package or meta-package itself. Message and service types define the structure of the data that is transmitted over ROS.

Inside a package there might be different folders and two required files: *package.xml* which functionality is mentioned above and *CMakeList.txt* which is necessary to compile and build the package dependencies. The different folders that can be found in different packages are *msg* with custom message definitions, *cfg* with parameter definition files, *launch* which stores executable files, *src* which stores all files of the main codes, that is node source code files, *srv* with service definition files, et cetera.

2.3.2 ROS Computation Graph Level

This level covers the basic elements of the ROS network each of which has its own data feed features. The elements of the present level are listed below:

- Nodes.
- Master.
- Parameter Server.
- Messages.
- Topics.
- Services.
- Bags.

A node is an independent and separately executable code that is responsible for execution of certain tasks, including hardware initialization or/and control, processing incoming data or operator signals, et cetera. All nodes are combined into one graph and communication among them is provided by means of ROS data streamers such as topics, services and the Parameter Server. This approach provides additional fault tolerance since the error in one particular node will not lead to the collapse of the entire system. Furthermore, the process of code generation is simplified. ROS is designed in such a way that individual nodes that perform similar functions but written in different programming languages can easily be interchanged each other without debugging. (ROS 2012a). Typical node in ROS includes a number of APIs:

- Slave which receives callbacks from the Master and establishes connection with other slaves.
- Transmission Control Protocol for ROS (TCPROS) and User Datagram Protocol for ROS (UDPROS) through which the nodes establish connection and exchange data via topics.
- Command-line which enables names within a node to be configured. (ROS 2014c).

In most cases, the selected station from where the nodes are launched does not affect the code functions in ROS. The exception to this rule may be the driver nodes that are to be run from the station to which the wrapped equipment is connected. Regarding the traction system, the *epos_hardware* node which will be described further is run on the embedded industrial computer to initialize the EPOS controllers.

The Master is the central object of the entire ROS system which registers names, searches the necessary elements within the graph and establishes peer-to-peer communication between two nodes using a topic. The connection establishment algorithm, which is shown in Figure 2.11 as an example of setting up a connection, when the *Camera* node announces the topic *images* and the node *Image viewer* subscribes to it, looks as follows:

1. One node “advertises” the topic, that is notifies the Master about the publication of data on a separate topic.
2. The Master node is waiting for any mode which will subscribe to this topic.

3. After a subscription, the Master establishes a peer-to-peer connection between the nodes through which data is exchanged.

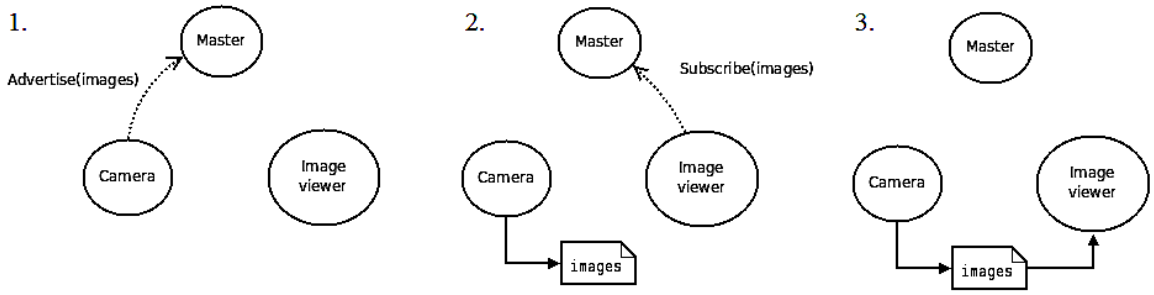


Figure 2.11. Algorithm for establishing connection by utilizing the Master node (ROS 2018a).

The Master is represented by a protocol on the basis of stateless Hypertext Transfer Protocol (HTTP) called Extensible Markup Language Remote Procedure Call (XML-RPC). This protocol was chosen due to the fact that it does not load the system heavily and does not require a state-full connection, which would track another device or connection over some time period.

The Master includes special APIs for registration of nodes as publishers, subscribers and services and is described by Uniform Resource Identifier (URI) which is bound to the *host:port* of the respective XML-RPC server and stored in the *ROS_MASTER_URI* environment variable. (ROS 2014c).

The initialization process of the Master node is started using the *roscore* command in the Ubuntu terminal of the main station which launches a ROS Master node, a ROS Parameter Server and a *rosout* logging node (ROS 2016c). In case of TIERA robot the station of which the Master node is launched is the robot station equipped with Advantech industrial computer (Poberezkin 2017). In Figure 2.12 one can see the example of *ros**core* command execution.

The data transfer within the graph, which, as mentioned above, unites all the nodes, is executed via simplest data structures, i.e. messages (ROS 2016b). Most of the data is transmitted through topics which represents a transport system of buses with anonymous

publish/subscribe semantics. Anonymity implies that, in most cases, nodes do not have the information about the node which published the message on the topic they are subscribed to, and, on the other side, topics that publish the message are not aware who their recipient is. In other words, data is transmitted through unidirectional many-to-many channels. (ROS 2014d).

```
efim@LUT6021:~$ roscore
... logging to /home/efim/.ros/log/e2b3507a-43a3-11e8-8f1f-000bab42059c
/roslaunch-LUT6021-2845.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://efim:44538/
ros_comm version 1.11.20


SUMMARY
========

PARAMETERS
 * /rosdistro: indigo
 * /rosversion: 1.11.20

NODES

auto-starting new master
process[master]: started with pid [2857]
ROS_MASTER_URI=http://efim:11311/

setting /run_id to e2b3507a-43a3-11e8-8f1f-000bab42059c
process[rosout-1]: started with pid [2870]
started core service [/rosout]
```

Figure 2.12. Execution of *roscore* command.

The main principle of the topics can be seen in Figure 2.13. It shows a graph that demonstrates some dependencies of *epos_hardware* package. The graph has been drawn using the special tool *rqt_graph* where the nodes are represented as ellipses and the topics are shown as rectangles. The nodes exchange data directly with each other avoiding the Master through the specified topic transport each of which has its own algorithm of sending and receiving messages (ROS 2014c).

In case, when the use of topics is not convenient for communication, for example, when “request-response” communication is required, another type of communication in ROS called “service” is used. Mainly used in distributed systems, service is represented by a

pair of messages and the process of communication itself is similar to a remote procedure call. (ROS 2012b). It should be mentioned that for each service there is only one provider which delivers data to a number of clients.

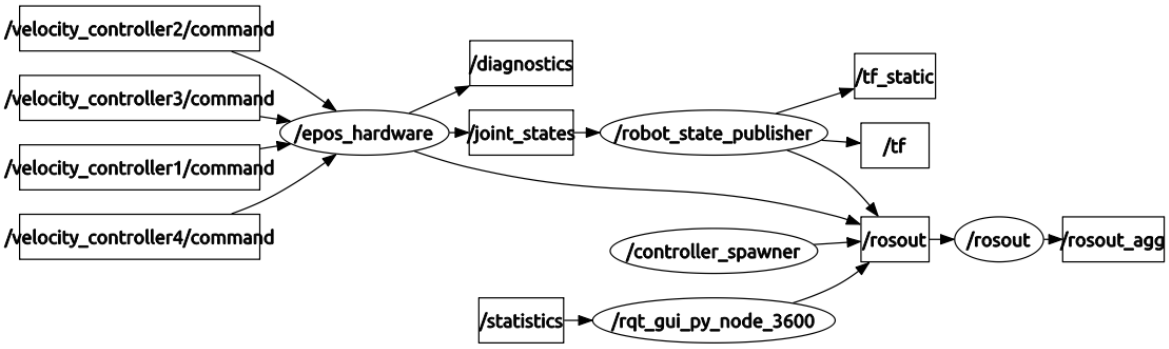


Figure 2.13. Dependencies of the part of *epos_hardware* package

The last but not least important tool for data exchange is the Parameter Server. As a rule, it is built into the Master and represented by the shared, multivariate dictionary that can be accessed using network APIs and each key of which represents a namespace. The Parameter Server is intended for storage of configuration parameters as ROS experiences difficulties when working with dynamic binary data. (ROS 2013). It is represented by XML-RPC as well due to which it is more flexible with data operations (ROS 2014c).

2.4 Communication system

The communication system of the TIERA robot was designed and implemented by the previous researcher Efim Poberezkin. It was developed taking into account that the robot should be tele-operated in conditions of aggressive environments (Poberezkin 2017). The robot utilizes wireless Wi-Fi network technology to exchange data between nodes which is provided by the hardware tools of the robot and two Robustel R3000-Q4LB industrial cellular routers which can be seen in Figure 2.14.

The characteristics of industrial routers can be found on the developer's website. Of the main characteristics worth noting their high reliability and the ability to establish connections using various network technologies such as 2G/3G/4G cellular standards or

802.11b/g/n WLAN interfaces which is utilized in TIERA robot (Robustel 2016). Due to LAN usage, each network station has its own IP.

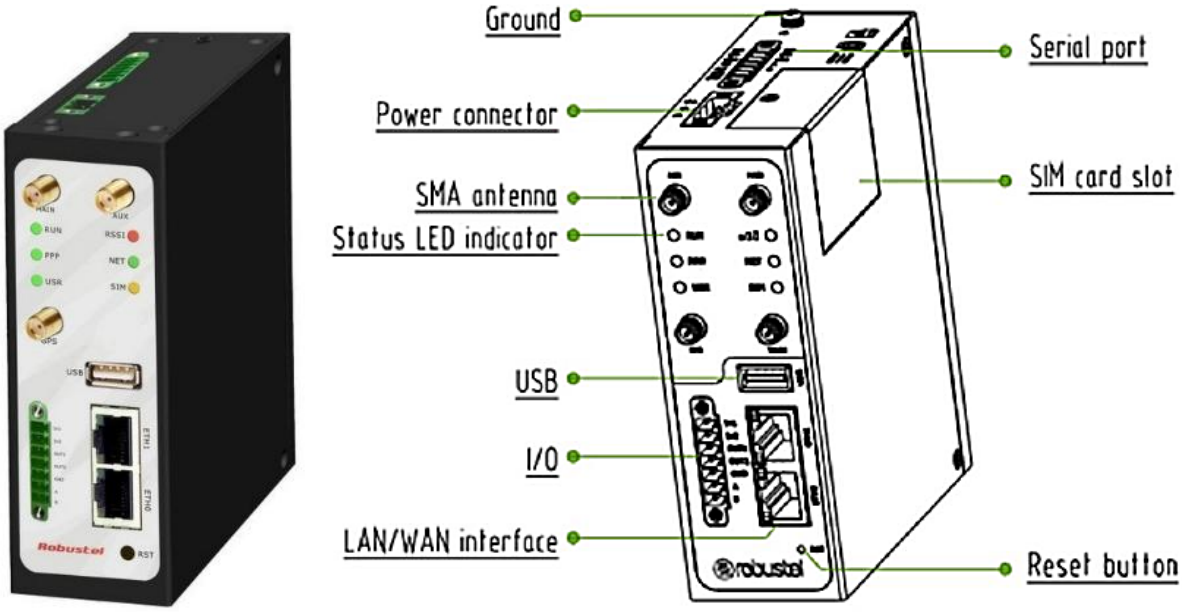


Figure 2.14. External view (left) and schematic layout of Robustel R3000-Q4LB Router (Robustel 2018).

More detailed information about the network can be learned from the Efim Poberezkin's master thesis (2017). The focus of the present thesis is not on the robot communication system and a brief overview which is provided in the present section should be sufficient.

3 DEVELOPMENT AND IMPLEMENTATION OF ROBOT TRACTION SYSTEM

Initially, this chapter will describe the software developed for the traction control systems which establishes connection between the robot station and hardware and allows the operator to control the robot position and trajectory in confined spaces. Furthermore, the problems that have arisen during the work on the project as well as the adjustments that have been applied to the system software in order solve them will be discussed. Afterwards, according to the requirement for remote control of the robot, the integration of all software packages into the robot and operator station will be described.

3.1 Developed software

As it was mentioned above, the robot traction system is based on ROS, for which a large number of user-contributed codes have been developed and stored on www.*github.com* website (GitHub 2015). The codes for initialization and control of the specific hardware that is utilized in the thesis project can be found there as well but in a more general form which requires further development. Therefore, since the LUT mobile robot utilizes four EPOS2 controllers instead of one, which is implied in the source code (GitHub 2015), it was necessary to adjust the code in order for the traction system of the TIERA robot to operate accurately and without errors.

The main packages developed and adjusted during the project are *epos_hardware* which performs the function of the motor wrapper and was initially developed by Mitchell Wills from RiverLab, Northeastern University (GitHub 2015) and custom developed *lut_track* which carries out the kinematics calculations shown above, allows one to integrate the Xbox controller into the ROS environment and provides high level control of the traction system. Both packages will be discussed in more detail below.

Below is a list of main codes and packages that are used to control the TIERA traction system. *epos_hardware* package consists of the following elements which are stored in the robot station and executed from it:

- epos_hardware
  - include
  - launch
    - *example_test_mcs.urdf*
    - *tiera_motor1.yaml*
    - *tiera_motor2.yaml*
    - *tiera_motor3.yaml*
    - *tiera_motor4.yaml*
    - *tiera_track.launch*
    - *tiera_track_robot.launch*
    - *tiera_track_station.launch*
  - src
    - nodes
      - *epos_hardware_node.cpp*
    - tools
      - *check_settings.cpp*
      - *get_state.cpp*
      - *get_state_rs232.cpp*
      - *list_devices.cpp*
    - util
      - *epos.cpp*
      - *epos_hardware.cpp*
      - *epos_manager.cpp*
      - *utils.cpp*
- epos_library

lut_track package, which should be stored and launched from the station to which the Xbox controller is connected, contains the following files and folders:

- include
- launch
- msg
  - *accurate_xy.msg*
  - *joy_xy.msg*
- srv
  - *kinematics.srv*
  - *kinematics_accurate.srv*
- src
  - *joy_node.cpp*
  - *kinematics_server.cpp*
  - *lut_controller.cpp*

3.1.1 epos_hardware package

Each file has its own specific functionality and is responsible for certain operations. For instance, *tiera_track.launch*, *tiera_track_robot.launch* and *tiera_track_station.launch* are the launch files which, as any file of this type, provide a procedure that allows one to run multiple ROS nodes at the same time (ROS 2018b) which are run using *roslaunch* command. Also it is worth noting that individual nodes can be also run in the ROS environment using the *rosrun* command. The commands given in the terminal are as follows:

$ *rosrun <package_name> <file.cpp* or *file.py> <possible parameters requested in the code* (optional)>

$ *roslaunch <package_name> <file.launch> <possible parameters requested in the code* (optional)>

tiera_track_robot.launch file serves as the motor wrapper which is initialized on the Advantech PC. It loads the robot description into the ROS workspace from the *example_test_mcs.urdf* file, executes *epos_hardware_node.cpp* which assigns the parameters from the Parameter Server to the motors, initializes the controller manager of

the joint state and velocity controllers for each motor and sets the name and other parameters to each of these hardware controllers.

tiera_track_station.launch file is initialized on the operator station and provides the necessary tools for steering wheels at a high level, i.e. it initializes the *joy* node which loads the Xbox controller which is connected to the operator station into the ROS workspace, runs the *lut_controller.cpp* node which outputs the high level controller to another terminal and executes *kinematics_server.cpp* which enables the kinematic calculation service.

Such division of the code functionality and the implementation of its parts at different stations allow one to control the mobile robot remotely. *tiera_track.launch* file represents the aforementioned two codes combined together which is designed to control the wheels using the Xbox joystick directly connected to the Advantech computer. Dependencies of *epos_hardware* and *lut_track* packages obtained using *rqt_plot* tool when both *tiera_track_station.launch* and *tiera_track_station.launch* files are executed can be found in appendix 3.

It is worthwhile to tell in more detail about the files that either have already been mentioned above, or listed as part of packages. *example_test_mcs.urdf* is a Universal Robot Definition File (URDF) which describes all connections and joints of the robot thus allowing the respective launch file to recognize the robot.

tiera_motor#.yaml files, where # represents motor numbers from 1 to 4 going counter-clockwise from the rear left wheel to the front left one, provide numerical values of the parameters for each motor which are declared in the package. Basic definition of each parameter is presented on the wiki page of the package (ROS 2015). Each parameter and actuator is assigned their particular hexadecimal number in the EPOS firmware and library space. The serial numbers of the motors which are essential for addressing parameters to them are as follows:

- motor 1: 0x662080006194;
- motor 2: 0x662080006193;

- motor 3: 0x662080006186;
- motor 4: 0x662080006192.

One of the most important files is *epos_hardware_node.cpp* which performs the function of the EPOS2 controller wrapper. It creates the *epos_hardware_velocity* node, stores all motor names passed through the *tiera_motor#.yaml* files loaded by the respective launch file that executes the node or command-line tool *rosrun* throughout argument of the motor serial number.

It creates the *epos_hardware::EposHardware* robot instance which passes the public node handle, the private node handle and the motor names into the ROS workspace. It also generates the instance of *controller_manager::ControllerManager* class type which passes the *epos_hardware::EposHardware* robot object and public node handle and introduces ROS interface for loading, starting, stopping and unloading EPOS2 and other ROS-operated controllers. Furthermore, it serializes execution of all running controllers during update. The inheritance diagram for *epos_hardware::EposHardware* class can be observed in Figure 3.1.

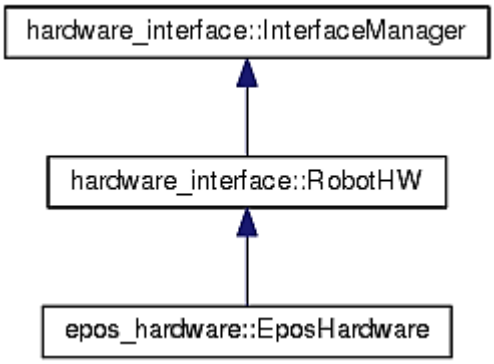


Figure 3.1. Inheritance diagram for *epos_hardware::EposHardware* class (ROS 2016d)

The *epos_hardware::EposHardware* class member functions such as *init, read, update_diagnostics, write* are declared in the *epos_hardware.cpp* file. *epos.cpp, epos_hardware.cpp* and *epos_manager.cpp* are *Epos::Epos, EposHardware:: EposHardware* and *EposManager::EposManager* class definitions, respectively, which are used in the internal code of the program as a way of storing and using the basic

functions of the EPOS library. *utils.cpp* defines some function definitions related to the EPOS library.

Afterwards, the *epos_hardware_node.cpp* defines a single thread for ROS services callback and makes a manual initialization of the node. It plots the *ROS_INFO* message about initialization of the motors and deploys the *ROS_FATAL* message of "Failed to initialize motors" in case if the *epos_hardware::EposHardware* robot object cannot be initialized. In case of successful initialization it displays the message "Motors Initialized", sets the controller rate to 50 Hz and storages the current ROS time in the variable.

In a loop, when there are no errors in ROS, the system reads joint states from the hardware, then the *controller_manager* object updates the iteration passing the time of the present and previous iterations and updated joint states of the robot and updates the previous variable that stores time to the current iteration time for next loop. Finally, the diagnostics tools are updated and the controller remains in standby mode until the next iteration is performed through time indicated in *controller_rate* variable. One can see dependencies of the *epos_hardware* package in appendix 4 which was obtained using special ROS utility called *rospack graph*.

Besides, in order to assess the state of the system and to check for errors special tools such as *check_settings.cpp*, *get_state.cpp*, *get_state_rs232.cpp* and *list_devices.cpp* have been introduced into the package. *check_settings.cpp* tool provides the data about all four motors, including motor serial number, operation mode, motor type, maximum following error, maximum profile velocity, maximum acceleration, position value, velocity value, current value, values of the regulator coefficients and digital output values of controllers, which is stored in the Parameter Server.

An example of using this utility when the connection is established via RS-232 interface at the baudrate of 115 kbit/s is shown in Figure 3.2. The characteristics are given only for motor 3. As can be seen from it special "error logs" were introduced into this tool. This solution will be explained in greater detail below.

get_state.cpp and *get_state_rs232.cpp* are utilized to obtain information about the separate EPOS2 controller and Maxon motor. The first utility is used when the connection is established via USB and the second one – when the Advantech PC and the master EPOS2 controller are connected via RS-232.

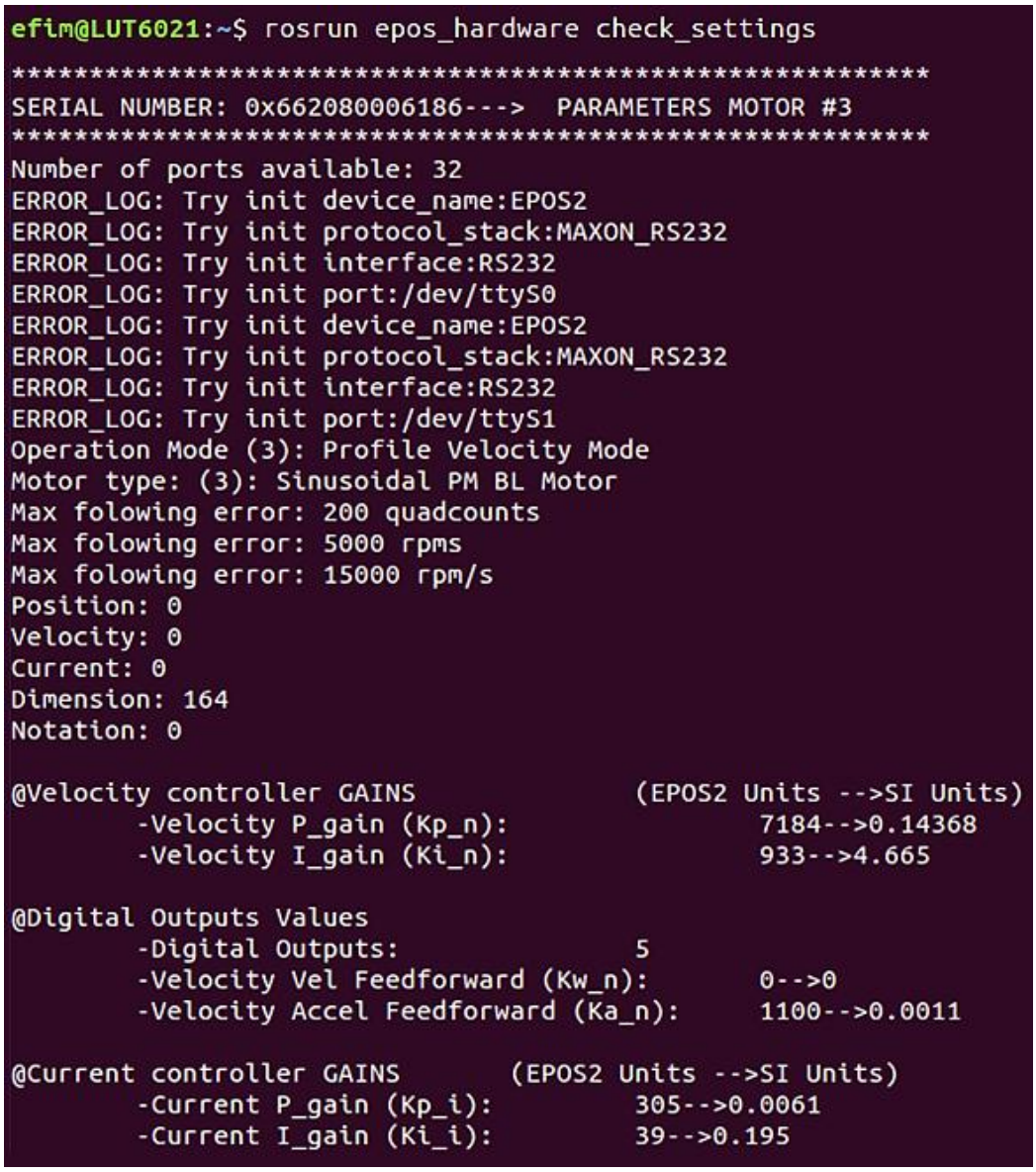


Figure 3.2. Executing *check_settings.cpp* tool.

list_devices.cpp tool is designed to detect the connection to the motors via various interfaces such as USB, RS-232 at different baud rate configurations or CANopen and make a list of them providing their serial numbers at the output of the terminal window. Figure 3.3 shows an example of the execution of abovementioned program.

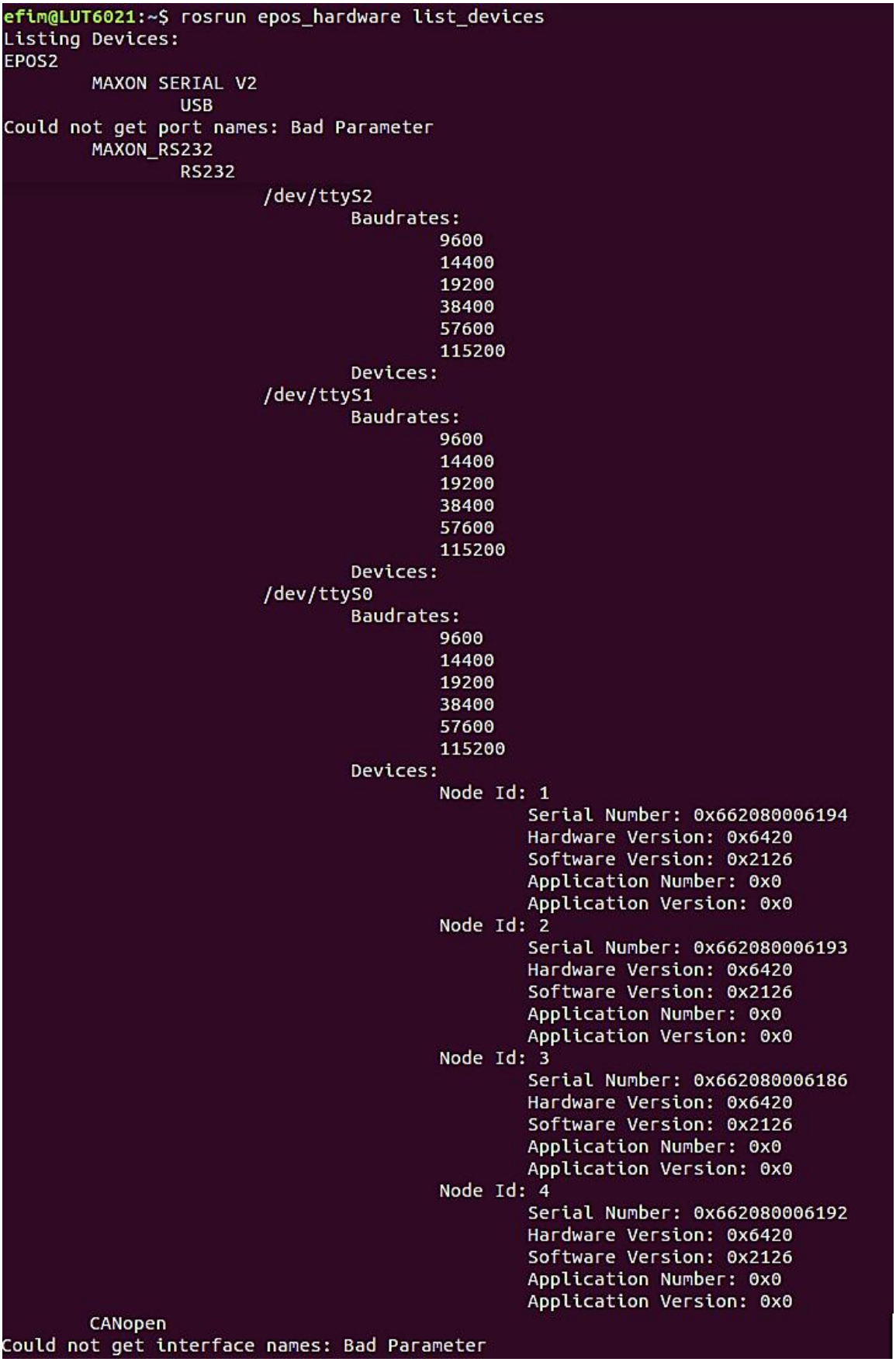


Figure 3.3. Executing *list_devices.cpp* tool.

3.1.2 lut_track package

The other package which is called *lut_track* includes the following files: custom message files *accurate_xy.msg* and *joy_xy.msg* for tracing specific topic definitions, service files *kinematics.srv* and *kinematics_accurate.srv* which defines velocity in X-Y coordinate system and yaw rate as inputs of the robot and wheel angular velocities ω_1, ω_2, ω_3 and ω_4 as its outputs and *joy_node.cpp* node which enables Xbox joystick in the ROS workspace. Special attention should be paid to the *kinematics_server.cpp* which calculates kinematics of the robot traction system and *lut_controller.cpp* which carries out high level control of the traction system.

kinematics_server.cpp file is a service node that performs inverse and direct kinematics calculations which are described in chapter 2.2. This file is arranged in such a way that it receives the necessary data about the motors and required trajectory from the ROS workspace, arranges them in the form of matrices, produces the necessary mathematical calculations and returns data. The signs of the wheel rotation vectors are presented in Table 2. Figure 3.4 depicts the directions of wheel rotation necessary for movement of the robot in a certain trajectory which are given in Table 2.

Table 2. Signs of the wheel rotation vectors.

Direction	**Wheel 1**	**Wheel 2**	**Wheel 3**	**Wheel 4**
Forward	$+\omega_i$	$-\omega_i$	$-\omega_i$	$+\omega_i$
Backward	$-\omega_i$	$+\omega_i$	$+\omega_i$	$-\omega_i$
Left	$+\omega_i$	$+\omega_i$	$-\omega_i$	$-\omega_i$
Right	$-\omega_i$	$-\omega_i$	$+\omega_i$	$+\omega_i$
Left twist	$-\omega_i$	$-\omega_i$	$-\omega_i$	$-\omega_i$
Right twist	$+\omega_i$	$+\omega_i$	$+\omega_i$	$+\omega_i$
Forward+left ($\omega_{ia}>>\omega_{ib}$)	$+\omega_{ia}$	$-\omega_{ib}$	$-\omega_{ia}$	$+\omega_{ib}$
Forward+right ($\omega_{ia}<<\omega_{ib}$)	$+\omega_{ia}$	$-\omega_{ib}$	$-\omega_{ia}$	$+\omega_{ib}$
Backward+left ($\omega_{ia}<<\omega_{ib}$)	$+\omega_{ia}$	$+\omega_{ib}$	$-\omega_{ia}$	$-\omega_{ib}$
Backward+right ($\omega_{ia}>>\omega_{ib}$)	$+\omega_{ia}$	$-\omega_{ib}$	$-\omega_{ia}$	$+\omega_{ib}$

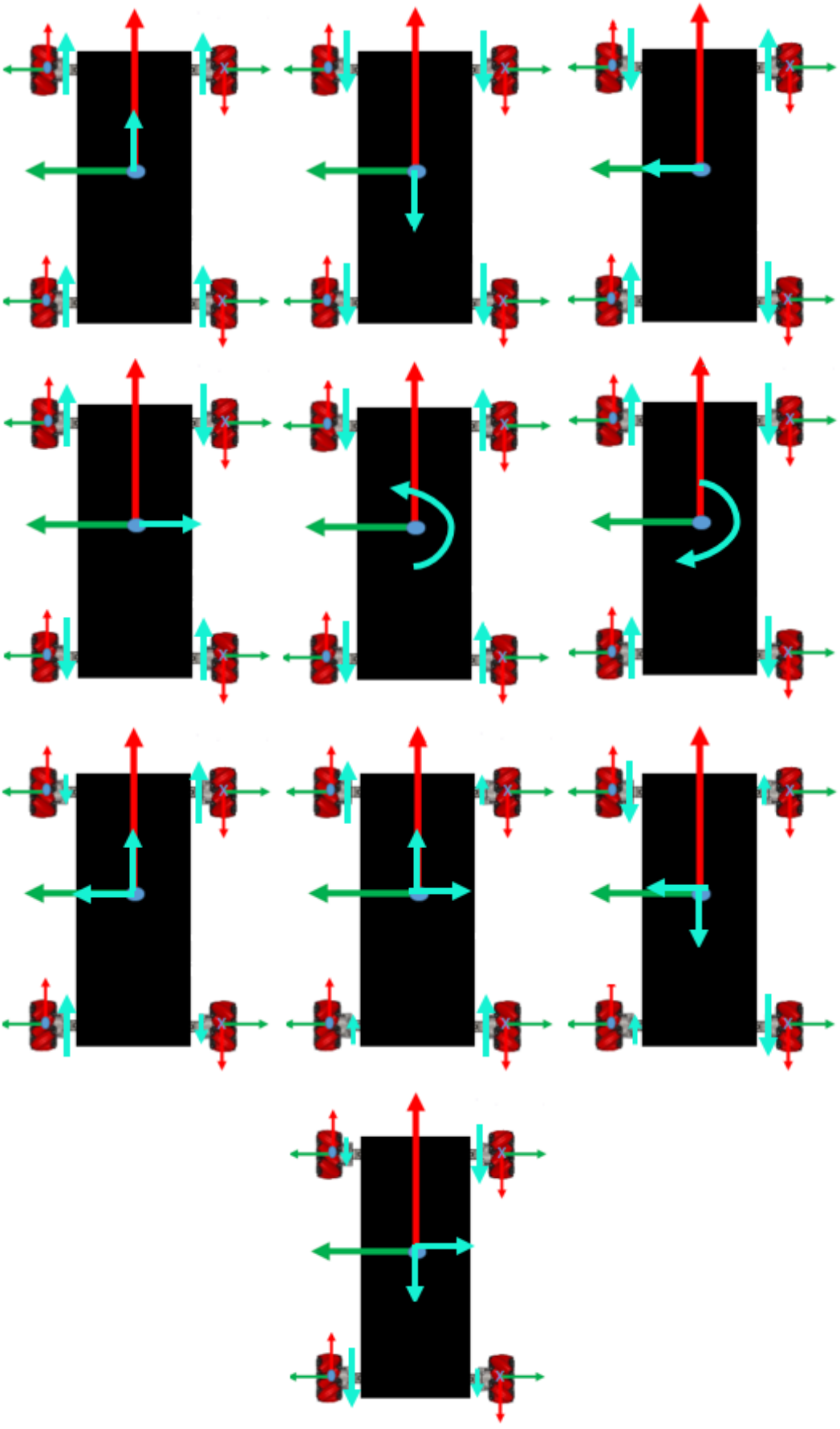


Figure 3.4. Directions of the Mecanum wheel rotations.

As established above, *lut_controller.cpp* file generates the respective node and construct the *lut_controller* object which carries out a high level control. As long as there are no errors in the ROS, it expects for updates on callbacks which will declare the necessary operations. The list of the topics with their respective callbacks, which the node is subscribed to, can be found in Table 3.

Table 3. Subscribed topics.

Topic name	Message type	Callback
/stop_status	*Std_msgs::Float64*	*LutMars::controller_callback*
/joy	*Sensor_msgs::Joy*	*LutMars::joyCallback*
/connection_state_alert	*Std_msgs::Int8*	*LutMars::ping::callback*
/normal_mode	*Lut_track::joy_xy*	*LutMars::normal_callback*
/accurate_mode	*Std_msgs::accurate_xy*	*LutMars::accurate_callback*

Every callback mentioned in Table 3 refers to private classes of the node and has its functions. *controller_callback* storages updates of the Xbox controller stop states. For instance, velocity calculation is enabled if "1" is set in *state_test* variable and velocities of all the wheels are set to zero if "0" is set. *joyCallback* is responsible for updating all variables associated with direct signals coming from the Xbox controller. For example, it sets current time of execution and establishes the time step between joystick updates, creates stored in the */joy* topic which is transfer to the kinematics server, et cetera. *ping::callback* provides the safety measures and sets the robot velocity to zero if there is no ping between the robot and operator stations.

In order to control the robot normal and accurate control modes have been introduced. Normal mode has been created to ensure maximum speed when moving in relatively open spaces and *normal_callback*, which is responsible for many functions, including creating of a service object to make a call to the kinematics server, sending command to all wheels in rpm, et cetera, is triggered when this mode is set. During operation in normal mode, the system checks if the absolute values of velocity are lower or equal to 5% of the maximum value which can be set on the joystick. If this condition is not met then the kinematics calculation tool works in the nominal mode. Otherwise the robot speed is set to zero.

Accurate mode is utilized when greater precision of movements is required. In this mode, the time which one need to hold the joystick in the desired direction is increased in order to avoid accidental movement of TIERA. The maximum speed of wheel rotation is reduced from 40 to 9 rpm compared to the normal mode. *accurate_callback*, which provides the same functionality as *normal_callback* for normal mode, is triggered when the accurate mode is set.

After giving the kinematics server command about the motion mode and subsequently receiving the values of the rotation speeds from it, the node publishes them for each controller as a separate topic. The list of the published topics can be found in Table 4. More detailed scheme of *lut_track* package can be seen in appendix 5.

Table 4. Published topics.

Topic name	Message type	Publisher
/velocity_controller1/command	*Std_msgs::Float64*	*Vel_pub_1*
/velocity_controller2/command	*Std_msgs::Float64*	*Vel_pub_2*
/velocity_controller3/command	*Std_msgs::Float64*	*Vel_pub_3*
/velocity_controller4/command	*Std_msgs::Float64*	*Vel_pub_4*
/target_marker	*Geometry_msgs::Point*	*Target_marker*
/stop_status	*Std_msgs::Float64*	*Stop_flager*
/normal_mode	*Lut_track::joy_xy*	*Normal_xy_pub*
/accurate_mode	*Std_msgs::accurate_xy*	*Accurate_xy_pub*

3.2 Software and hardware troubleshooting

During the development and further implementation of the software, there occurred issues associated with the features of the utilized hardware. Following two cases that may be of particular interest in terms of the robot development using identical equipment will be described further.

3.2.1 USB and RS-232 interfaces

According to Feature Chart of the EPOS controllers, the optimal way to connect the master controller with an industrial computer is the USB protocol as it provides the highest data transfer rate compared to RS-232 and CANopen protocols: 12 Mbps, 115 kbit/s and

1 Mbit/s, respectively (Maxon 2016b). While working on the project it was established that it was possible to detect the motors and control each motor individually utilizing *epos_hardware* code in ROS. Moreover, when trying to control all wheels at the same time it was possible to send commands to all of them at the beginning of the robot operation. Nevertheless, USB-communication with the wheels was lost afterwards. Besides, the motors continued to execute the command, which they received before the connection broke down, without reacting to further commands from the operator.

As it was established, RAM of the EPOS2 70/10 controllers is limited and they can process restricted number of commands at the same time. If USB protocol detects that some sent packages are dropped it resends the information until the whole data package is received. Thus, if the memory of the EPOS2 is overflowed by extensive number of commands which the master EPOS2 controller fails to transfer to the slaves using the CANopen protocol in due time, there is a probability of a closed loop on the USB data circuit that will block any other interactions or commands from the operator to the controllers and motors with wheels, respectively. Figure 3.5 depicts the USB communication algorithm while sending and receiving data frames.

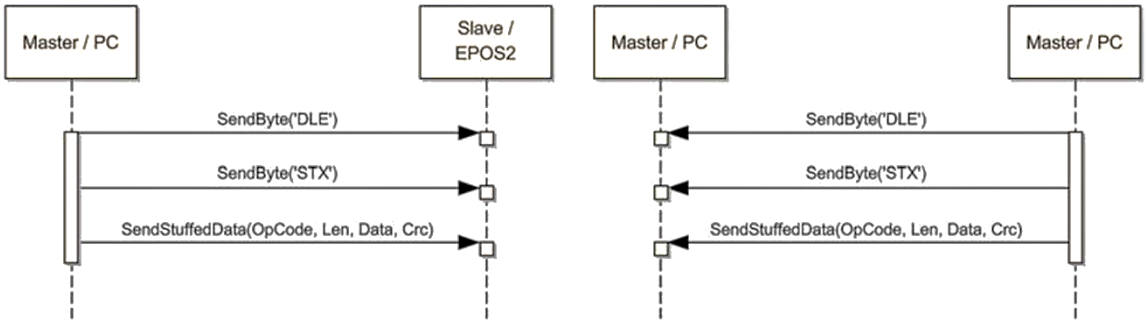


Figure 3.5. Sending and receiving data frames utilizing USB communication (Maxon 2016c).

The reason which can proof the abovementioned assumption is that only one EPOS (#1, rear left wheel) acts as the master node which receives the commands for all four motors from the Advantech PC and then transmit them through the CANopen network. Due to the difference in the data transfer rate between the CANopen and USB protocols, the RAM of

the first EPOS2 controller gradually filled up as it did not have time to transfer all data to subsequent series-connected controllers.

When working with one motor, no problems with receiving and processing code and commands were uncovered. However, this bug occurs when controlling more than one motor. Since the RS-232 protocol does not require a feedback response on receiving all data packages (Maxon 2016c) and the data transfer rate of this protocol correlates with the one of CANopen, it was decided to implement it instead of USB in order to establish connection between the robot PC and EPOS2 controller. One can observe algorithm for sending and receiving data frames using RS-232 communication in Figure 3.6.

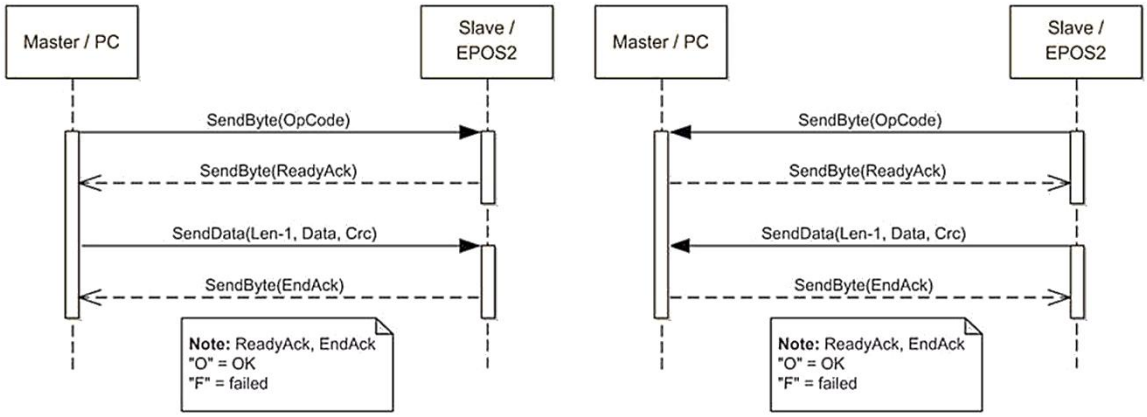


Figure 3.6. Sending and receiving data frames utilizing RS-232 communication (Maxon 2016c).

Nevertheless, when trying to work with the robot PC, the serial port for RS-232 had to be configured in order to be detected. This problem was encountered when the code was implemented to the robot station because the Advantech PC is run on Ubuntu only. Consequently, its ports have to be configured by the traction system software to avoid errors. Figure 3.7 depicts the problem with the EPOS2 controllers which occurred on the robot station.

When working on VirtualBox installed in Windows host, the serial port is automatically detected and configured by Windows OS which passes the settings directly to VirtualBox. Moreover, as can be seen from the description of the Advantech PC provided above, it has

several serial ports unlike a conventional computer that has only one RS-232 connector which complicates the task of perceiving the code for the robot station.

```
[INFO] [WallTime: 1524124301.892728] Controller Spawner: Waiting for service controller_
manager/load_controller
[INFO] [WallTime: 1524124301.894716] Controller Spawner: Waiting for service controller_
manager/switch_controller
[INFO] [WallTime: 1524124301.896588] Controller Spawner: Waiting for service controller_
manager/unload_controller
[INFO] [WallTime: 1524124301.898724] Loading controller: joint_state_controller1
[INFO] [WallTime: 1524124301.966996] Loading controller: velocity_controller1
[INFO] [WallTime: 1524124302.045302] Loading controller: joint_state_controller2
[INFO] [WallTime: 1524124302.051491] Loading controller: velocity_controller2
[INFO] [WallTime: 1524124302.060928] Loading controller: joint_state_controller3
[ERROR] [1524124302.062601046]: Could not find motor
[ERROR] [1524124302.062673994]: Could not configure motor: my_wheel_actuator1
[INFO] [WallTime: 1524124302.066894] Loading controller: velocity_controller3
[INFO] [WallTime: 1524124302.082117] Loading controller: joint_state_controller4
[INFO] [WallTime: 1524124302.088275] Loading controller: velocity_controller4
[INFO] [WallTime: 1524124302.098487] Controller Spawner: Loaded controllers: joint_state
_controller1, velocity_controller1, joint_state_controller2, velocity_controller2, joint
_state_controller3, velocity_controller3, joint_state_controller4, velocity_controller4
[ERROR] [1524124302.891328261]: Could not find motor
[ERROR] [1524124302.891382850]: Could not configure motor: my_wheel_actuator2
[ERROR] [1524124303.714310003]: Could not find motor
[ERROR] [1524124303.714366572]: Could not configure motor: my_wheel_actuator3
[ERROR] [1524124304.529825523]: Could not find motor
[ERROR] [1524124304.530040342]: Could not configure motor: my_wheel_actuator4
[FATAL] [1524124304.530117966]: Failed to initialize motors
```

Figure 3.7. Error when initializing EPOS2 controllers on the robot station.

One of Ubuntu OS features is that the system contains a list of serial ports for the RS-232 interface which consists of 32 names only 3 of which represent a real physical port (see Advantech PC description). As it was found later, the EPOS2 wrapper, that is *epos_hardware* package, does not initialize the connection between the computer and the controllers via real serial ports but only through nonexistent ones. In order to solve the issue the code of the package has been changed. In particular, when the initialization file is executed, the port list is cleared and the connection between the robot PC and EPOS2 controllers through existing physical serial ports for the RS-232 interface is forced via *utils.cpp* file. The code of this node is given in appendix 6.

Despite the disadvantages of this method, such as the need to manually declare the ports using code, this solution was found to be optimal because it allows one to directly address the hardware. In addition, these port names are identical for the same OS, that is, the difference can be generally observed in the number of ports. In order to avoid the abovementioned problems in the future, special “error logs” which allow one to quickly

identify failure causes and to choose the problem solving technique have been also introduced into the code.

3.2.2 Magnetic safety brakes

In order to prevent any unintentional movement when the controller is disabled the traction system is equipped with the safety or holding brakes. One of the detected problems was locking of the safety brakes of the motor 1 (the master of the CANopen network) when the EPOS2 controllers were initialized via ROS software.

As it was established from extensive reading of EPOS2 documentation, in order to force the release of the brakes, the digital outputs of the controller are to be configured (Maxon 2017). This is done by adding special edits to the code that are specified in the Command Library documentation (Maxon 2016e). In order to avoid similar brake lock-up cases with other controllers, this technique was also applied to the rest three slave EPOS2 controllers.

EPOS2 70/10 controller is equipped with sixteen digital outputs, i.e. their values in decimal form can vary from 0 to 65535. When the read values from digital outputs are translated into binary form, one can establish which values are displayed on each separate digital output. Figure 3.8 shows the configuration of EPOS2 digital outputs by default presented in the official firmware documentation.

According to the documentation recommendations for the EPOS2 70/10 controller the functionality of the Holding Brake bit responsible for changing the state of the safety brakes has been assigned to the digital output 4 (0x2079-4) due to output current limitations (Maxon 2017). The brakes are unlocked when the configured digital output is in the active state, its mask, which allows one to modify the configuration of the output when its bit is set to “1” in this register, is enabled and low polarity, which means that the associated output is inverted, is established.

The code of *epos.cpp* file which configures a variety of motor parameters including the digital outputs is demonstrated in appendix 7. Particular attention should be paid to pages 15 and 16 where the part of the code, by means of which the brakes are unlocked, can be observed. *tiera_motor1.yaml* file which provides the values of these parameters for the first

motor can be observed as an example of such files in appendix 8 as the other three motors have the same configuration principle.

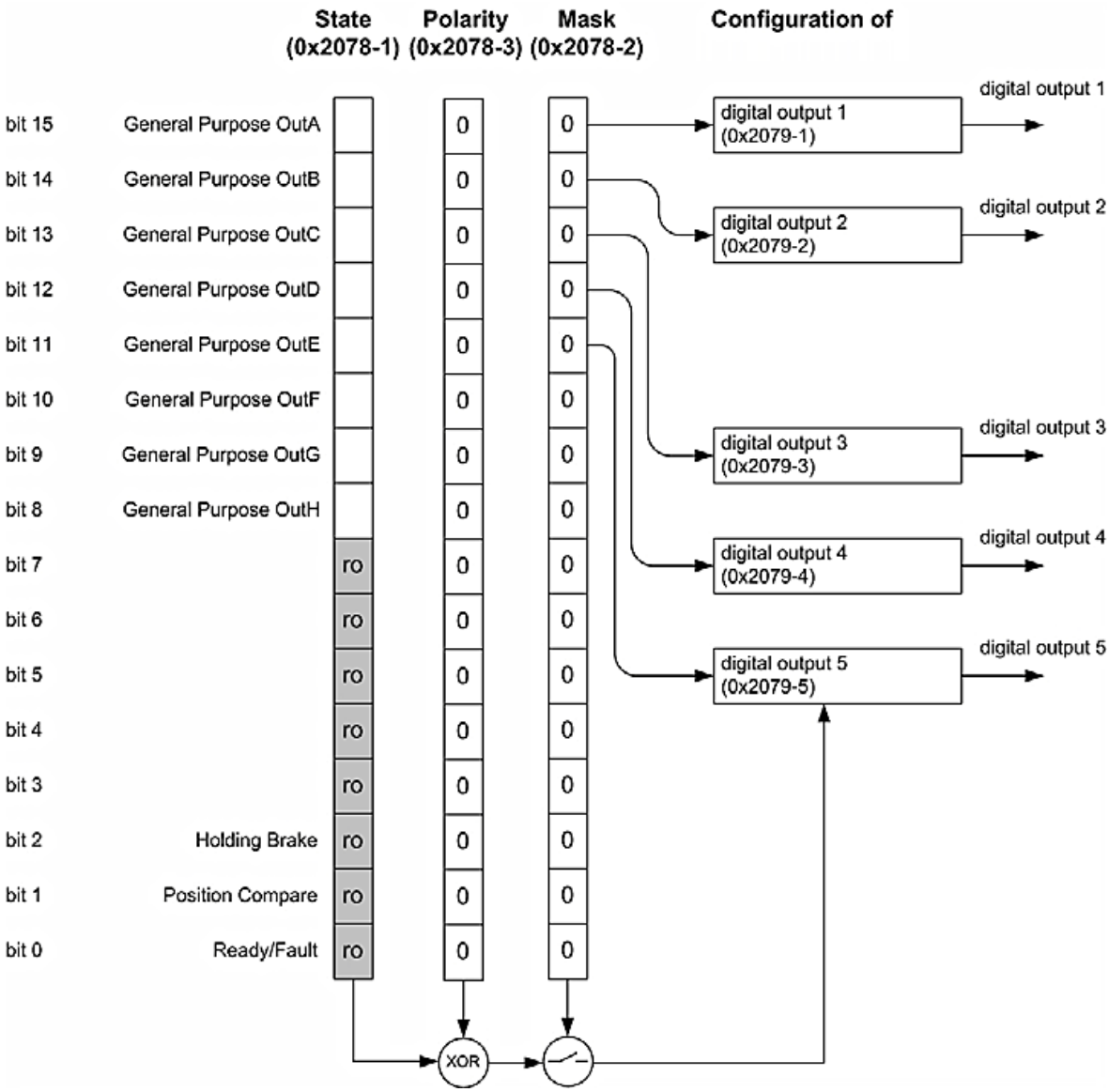


Figure 3.8. Configuration of EPOS2 digital outputs by default (Maxon 2017).

3.3 Remote control through SSH tunnel

As it has been mentioned the robot model in ROS is configured through links and joints. In order to control them t_f package which allows the user to follow a number of coordinate frames over time and, thus, make inverse and direct transformation through the t_f graph is utilized. Therefore, the coordinates of the each wheel can be read directly from the respective motor. The t_f graph of the launched traction system is shown in Figure 3.9.

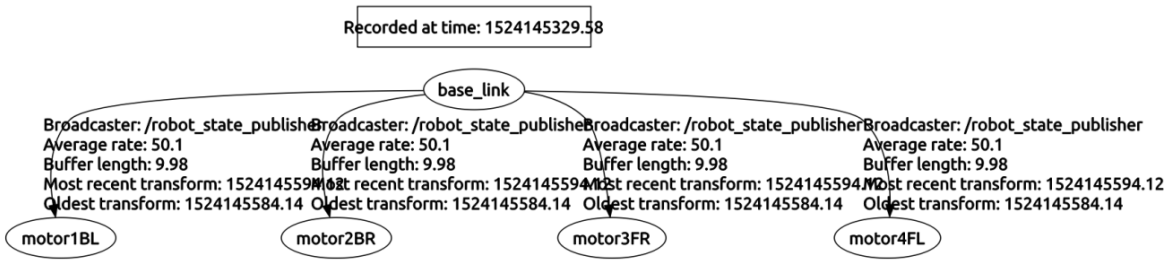


Figure 3.9. t_f graph of traction system.

One of the basic requirements for the TIERA mobile robot is the possibility of its remote control. After analyzing the possible options, it was decided to use the SSH network protocol which allows one to get remote access to a computer with a high degree of connection security.

Basically, SSH is implemented as two applications – the SSH server and the SSH client. Ubuntu is equipped with the *OpenSSH* software which performs the functions of the SSH client and server. When connecting, the client undergoes the authorization procedure at the server and an encrypted connection is established between them. In few words, SSH provides the ability to remotely execute commands and copy files with client and server authentication and encryption as well as compression of the transmitted data. It should be mentioned that passwords are also encrypted when utilizing this technique. (SSH 2017).

The most secure entry is provided with the key file which is protected with a password and, in most cases, this option is enabled on the server side. On the server and client machines, that are the operator and robot station, respectively, the passwords were created by Efim Poberezkin as a means of protection against hacking long before the creation of the SSH channel (2017). After all the configuration settings are complete, the following command is sent from the operator station to access the robot terminal:

$ *ssh user@host*

According to Poberezkin (2017), each element of the TIERA communication system (Wi-Fi routers of both operator and robot stations, the grippers of TIERA, et cetera) was assigned its own specific static IP address. For ease of use and development, new host

names were given to the IP addresses which allow one to replace them with the names when entering commands. Figure 3.10 shows the list of names for which the corresponding IP addresses were replaced.

Thus, taking into account new names for access to the robot station, it is necessary to enter the following command with the corresponding password:

$ *ssh efim@192.168.2.3* or
$ *ssh efim@efim*

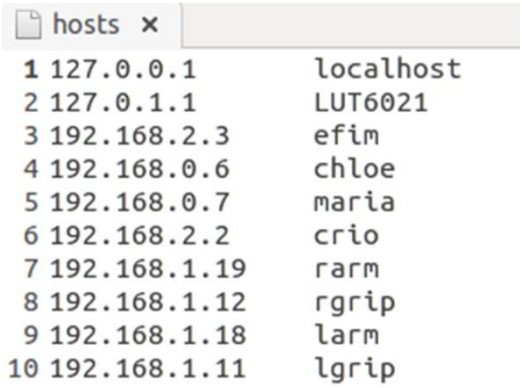


Figure 3.10. List of host names (Poberezkin 2017).

After entering the password for the robot station, the operator is given access to the Advantech terminal under the name *efim@LUT6021*. Afterwards, as mentioned above, to initialize EPOS2 controllers and connect them to ROS workspace *tiera_track_robot.launch* file, which performs the function of the motor wrapper, is to be executed using SSH tunnel. This is done by entering the following command in the Ubuntu terminal window of *efim@LUT6021*:

$ *roslaunch epos_hardware tiera_track_robot.launch*

Figure 3.11 shows the part of the terminal window when the launch file is initialized on the robot PC. Particularly, the process of loading parameters from various *tiera_track.yaml* files can be seen below. The code of the launch file can be traced in appendix 9.

```
efim@LUT6021:~$ roslaunch epos_hardware tiera_track_robot.launch
... logging to /home/efim/.ros/log/f165e5a2-372c-11e8-8793-000bab42059c/roslaunch-LUT6021-5227.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://efim:39428/

SUMMARY
========

PARAMETERS
 * /epos_hardware/my_wheel_actuator1/actuator_name: wheel_actuator1
 * /epos_hardware/my_wheel_actuator1/clear_faults: True
 * /epos_hardware/my_wheel_actuator1/current_regulator/gain/i: 46
 * /epos_hardware/my_wheel_actuator1/current_regulator/gain/p: 248
 * /epos_hardware/my_wheel_actuator1/motor/ec_motor/max_output_current: 10.5
 * /epos_hardware/my_wheel_actuator1/motor/ec_motor/nominal_current: 9.84
 * /epos_hardware/my_wheel_actuator1/motor/ec_motor/number_of_pole_pairs: 1
 * /epos_hardware/my_wheel_actuator1/motor/ec_motor/thermal_time_constant: 33.9
 * /epos_hardware/my_wheel_actuator1/motor/type: 10
 * /epos_hardware/my_wheel_actuator1/operation_mode: profile_velocity
 * /epos_hardware/my_wheel_actuator1/outputs/conf_outputs_values/configuration: 12
 * /epos_hardware/my_wheel_actuator1/outputs/conf_outputs_values/digital_output_nb: 4
 * /epos_hardware/my_wheel_actuator1/outputs/conf_outputs_values/mask: 1
 * /epos_hardware/my_wheel_actuator1/outputs/conf_outputs_values/polarity: 0
 * /epos_hardware/my_wheel_actuator1/outputs/conf_outputs_values/state: 1
 * /epos_hardware/my_wheel_actuator1/outputs/outputs_values: 5
 * /epos_hardware/my_wheel_actuator1/position_profile/acceleration: 8000
 * /epos_hardware/my_wheel_actuator1/position_profile/deceleration: 9000
 * /epos_hardware/my_wheel_actuator1/position_profile/velocity: 5000
 * /epos_hardware/my_wheel_actuator1/position_regulator/feed_forward/acceleration: 834
 * /epos_hardware/my_wheel_actuator1/position_regulator/feed_forward/velocity: 25829
 * /epos_hardware/my_wheel_actuator1/position_regulator/gain/d: 2450
 * /epos_hardware/my_wheel_actuator1/position_regulator/gain/i: 7380
 * /epos_hardware/my_wheel_actuator1/position_regulator/gain/p: 1959
 * /epos_hardware/my_wheel_actuator1/safety/max_acceleration: 15000
 * /epos_hardware/my_wheel_actuator1/safety/max_following_error: 100
 * /epos_hardware/my_wheel_actuator1/safety/max_profile_velocity: 5000
 * /epos_hardware/my_wheel_actuator1/sensor/incremental_encoder/inverted_polarity: False
 * /epos_hardware/my_wheel_actuator1/sensor/incremental_encoder/resolution: 500
 * /epos_hardware/my_wheel_actuator1/sensor/type: 1
 * /epos_hardware/my_wheel_actuator1/serial_number: 0x662080006194
 * /epos_hardware/my_wheel_actuator1/velocity_profile/acceleration: 2000
 * /epos_hardware/my_wheel_actuator1/velocity_profile/deceleration: 2000
```

Figure 3.11. Executing *tiera_track_robot.launch* file.

Upon loading all parameters of the motor configuration in the parameter server, EPOS2 controllers are initialized. Figure 3.12 shows the terminal window during this operation via executing *tiera_track_robot.launch* file. The code of the launch file can be traced in appendix 10.

After successful initialization of the controllers, it is necessary to establish a connection between the Xbox joystick and ROS workspace at the operator station, i.e. VirtualBox with the Linux OS installed on it to which this file was moved. As mentioned above *tiera_track_station.launch* file is responsible for the executing of this operation. In Figure 3.13 one can see an example of running the file on the operator station. The file is run by entering the following command in the Ubuntu terminal window:

$ *roslaunch epos_hardware tiera_track_station.launch*

```
NODES
  /
    controller_spawner (controller_manager/spawner)
    epos_hardware (epos_hardware/epos_hardware_node)
    robot_state_publisher (robot_state_publisher/state_publisher)

ROS_MASTER_URI=http://efim:11311

core service [/rosout] found
process[epos_hardware-1]: started with pid [5245]
process[controller_spawner-2]: started with pid [5246]
process[robot_state_publisher-3]: started with pid [5247]
[INFO] [WallTime: 1522756606.548544] Controller Spawner: Waiting for service controller_manager/load_c
ontroller
[INFO] [WallTime: 1522756606.551030] Controller Spawner: Waiting for service controller_manager/switch
_controller
[INFO] [WallTime: 1522756606.553489] Controller Spawner: Waiting for service controller_manager/unload
_controller
[INFO] [WallTime: 1522756606.555567] Loading controller: joint_state_controller1
[INFO] [WallTime: 1522756606.623489] Loading controller: velocity_controller1
[INFO] [WallTime: 1522756606.701290] Loading controller: joint_state_controller2
[INFO] [WallTime: 1522756606.708632] Loading controller: velocity_controller2
[INFO] [WallTime: 1522756606.719239] Loading controller: joint_state_controller3
[INFO] [WallTime: 1522756606.725767] Loading controller: velocity_controller3
[INFO] [WallTime: 1522756606.738177] Loading controller: joint_state_controller4
[INFO] [WallTime: 1522756606.744684] Loading controller: velocity_controller4
[INFO] [WallTime: 1522756606.755885] Controller Spawner: Loaded controllers: joint_state_controller1,
velocity_controller1, joint_state_controller2, velocity_controller2, joint_state_controller3, velocity
_controller3, joint_state_controller4, velocity_controller4
[INFO] [WallTime: 1522756650.248806] Started controllers: joint_state_controller1, velocity_controller
1, joint_state_controller2, velocity_controller2, joint_state_controller3, velocity_controller3, joint
 state_controller4, velocity_controller4
```

Figure 3.12. Process of controller initialization.

```
user@VBubuntu:~$ roslaunch epos_hardware tiera_track_station.launch
... logging to /home/user/.ros/log/f165e5a2-372c-11e8-8793-000bab42059c/roslaunch-VBubuntu-3830.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://chloe:32977/

SUMMARY
========

PARAMETERS
 * /joy_node/coalesce_interval: 0.05
 * /joy_node/deadzone: 0.1
 * /joy_node/dev: /dev/input/js3
 * /rosdistro: indigo
 * /rosversion: 1.11.20

NODES
  /
    joy_node (joy/joy_node)
    kinematics_server (lut_track/kinematics_server)
    lut_controller (lut_track/lut_controller)

ROS_MASTER_URI=http://efim:11311

core service [/rosout] found
process[joy_node-1]: started with pid [3839]
process[lut_controller-2]: started with pid [3840]
process[kinematics_server-3]: started with pid [3841]
[lut_controller-2] process has finished cleanly
log file: /home/user/.ros/log/f165e5a2-372c-11e8-8793-000bab42059c/lut_controller-2*.log
[kinematics_server-3] process has finished cleanly
log file: /home/user/.ros/log/f165e5a2-372c-11e8-8793-000bab42059c/kinematics_server-3*.log
```

Figure 3.13. Executing *tiera_track_station.launch* file.

In case the Xbox controller is damaged or inaccessible at the operator station each wheel can be launched directly by publishing messages at the topics through the *rostopic* command which addresses directly to EPOS2 controllers. The commands in general form and for the first wheel are as follows:

$ *rostopic pub* (publish information) *<topic name> <message type> <message value>*
$ *rostopic pub /velocity_controller1/command std_msgs/Float64 ’10’*

To start working with the robot traction system, it is necessary to switch the enable state on the Xbox joystick. To perform this operation guide button of the Xbox joystick which location can be seen in Figure 3.14 has been assigned. It sets “1” to the *state_test* internal variable switching the kinematics calculation tools to the active mode what was discussed in detail above.

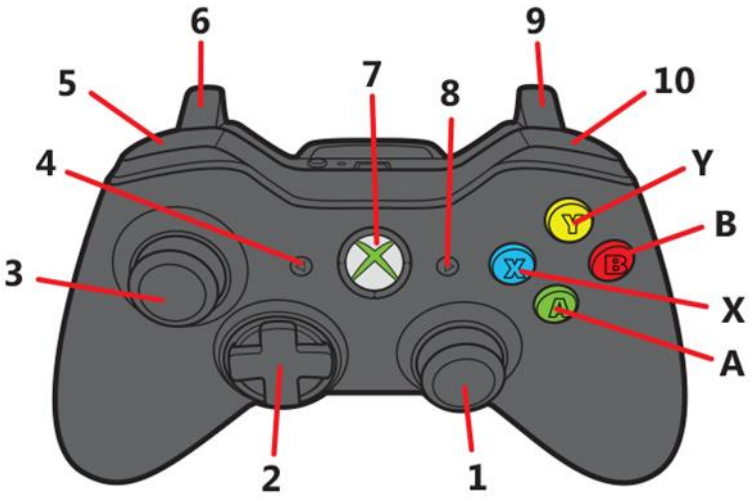


Figure 3.14. Xbox controller layout and button names: 1 – right stick, 2 – directional pad (D-pad), 3 – left stick, 4 – “back” button, 5 – left bumper, 6 – left trigger, 7 – guide button, 8 – “start” button, 9 – right trigger, 10 – right bumper, A – A-button (green), B – B-button (red), X – X-button (blue), Y – Y-button (yellow) (Xbox 2018).

The wheels of the TIERA robot are controlled with the left stick and the right and left triggers in the normal mode and with the directional pad and the right and left bumpers in the accurate mode. The normal operation mode is activated with the “back” button and the accurate operation mode – with the “start” button.

According to the structure of the code used in the traction system of the TIERA robot, the response rate, as well as the maximum speed of the robot, are controlled and adjusted by *tiera_motor#.yaml* files. Pre-configuration of the PID coefficients for position, velocity and current regulators embedded into EPOS2 can be carried out using special software such as EPOS Studio through a special utility.

4 RESULTS

Upon completion all the above actions the robot is ready for operation. Robot movement monitoring is carried out using a special terminal depicted in Figure 4.1 which displays the velocity of the robot, in general, and each wheel, in particular, as well as vision system developed by Igor Soroka (2016) and sonar system developed by Ekaterina Menshova (2017) in LabVIEW software and light detection and ranging (LiDAR) system developed by Henri Kaupilla in ROS.

```
PING: Active (1)
 ENABLING SYSTEM STATE (1)
OPERATION_MODE: NORMAL (1)
[ INFO] [1524321993.035837490]: Command 1: -19.217443
[ INFO] [1524321993.035906276]: Command 2: -19.217443
[ INFO] [1524321993.035930257]: Command 3: 19.217443
[ INFO] [1524321993.035949656]: Command 4: 19.217443
n1:-19.2174
n2:-19.2174
n3:19.2174
n4:19.2174
Velocity is -19.2174
Velocity changed to -19.2174

PING: Active (1)
 ENABLING SYSTEM STATE (1)
OPERATION_MODE: NORMAL (1)
[ INFO] [1524321993.105547968]: Command 1: -19.635595
[ INFO] [1524321993.105663588]: Command 2: -19.635595
[ INFO] [1524321993.105739471]: Command 3: 19.635595
[ INFO] [1524321993.105828303]: Command 4: 19.635595
n1:-19.6356
n2:-19.6356
n3:19.6356
n4:19.6356
Velocity is -19.6356
Velocity changed to -19.6356
```

Figure 4.1. Terminal window of the kinematics calculation tool.

The example of detecting various obstacles in a two-dimensional space with the LiDAR can be observed in Figure 4.2. Data received by the LiDAR is transmitted over ROS and visualized using a special utility called “rviz” at the operator station. The LiDAR provides 2D-image of the landscape at the height at which it is installed in the robot.

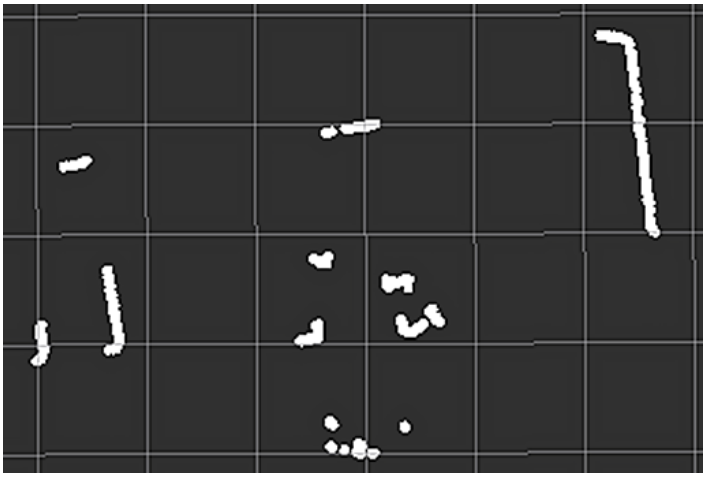


Figure 4.2. 2D-image received through the LiDAR.

To ensure the safety of the robot movement, it was equipped with eight sonars, two on each side. Three levels of security are displayed at the operator station screen depending on the distance to the obstacle on each sonar: green – 60 cm, yellow – 40 cm and red – 20 cm. In Figure 4.3 one can these levels displayed using specially developed LabVIEW program.

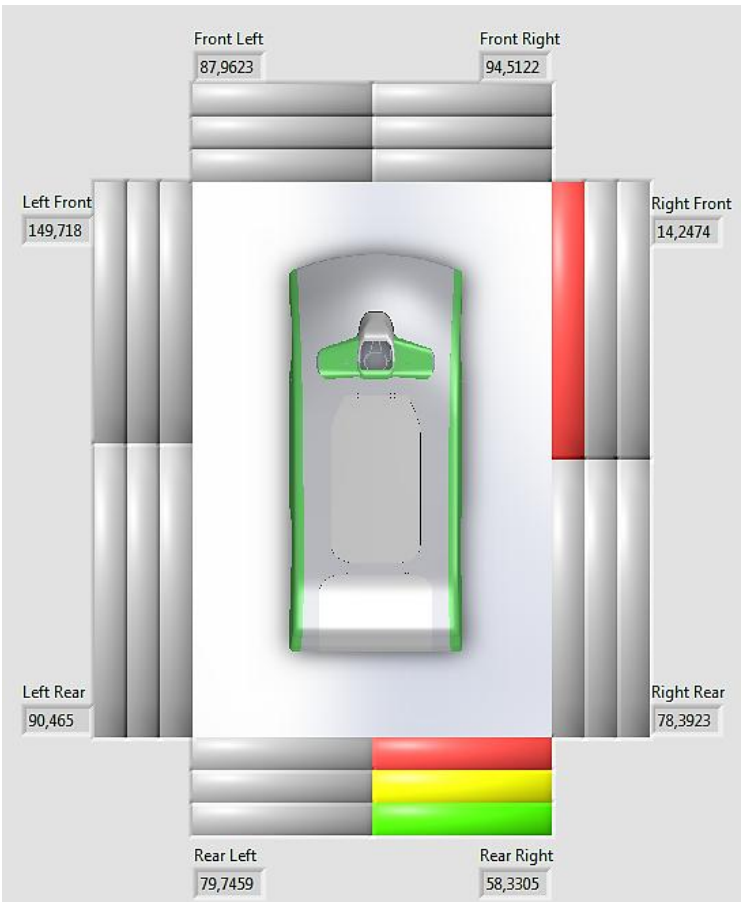


Figure 4.3. Identification of obstacles using sonars

The rotation speed of each wheel, as well as the commands sent to the velocity controllers can be tracked with *rqt_plot* tool which displays different scalar ROS variables. In Figure 4.5 one can observe the results of the test conducted in the Laboratory of Intelligent Machines. The velocity controller commands cannot be seen as they fully coincide with the joint states of each wheel.

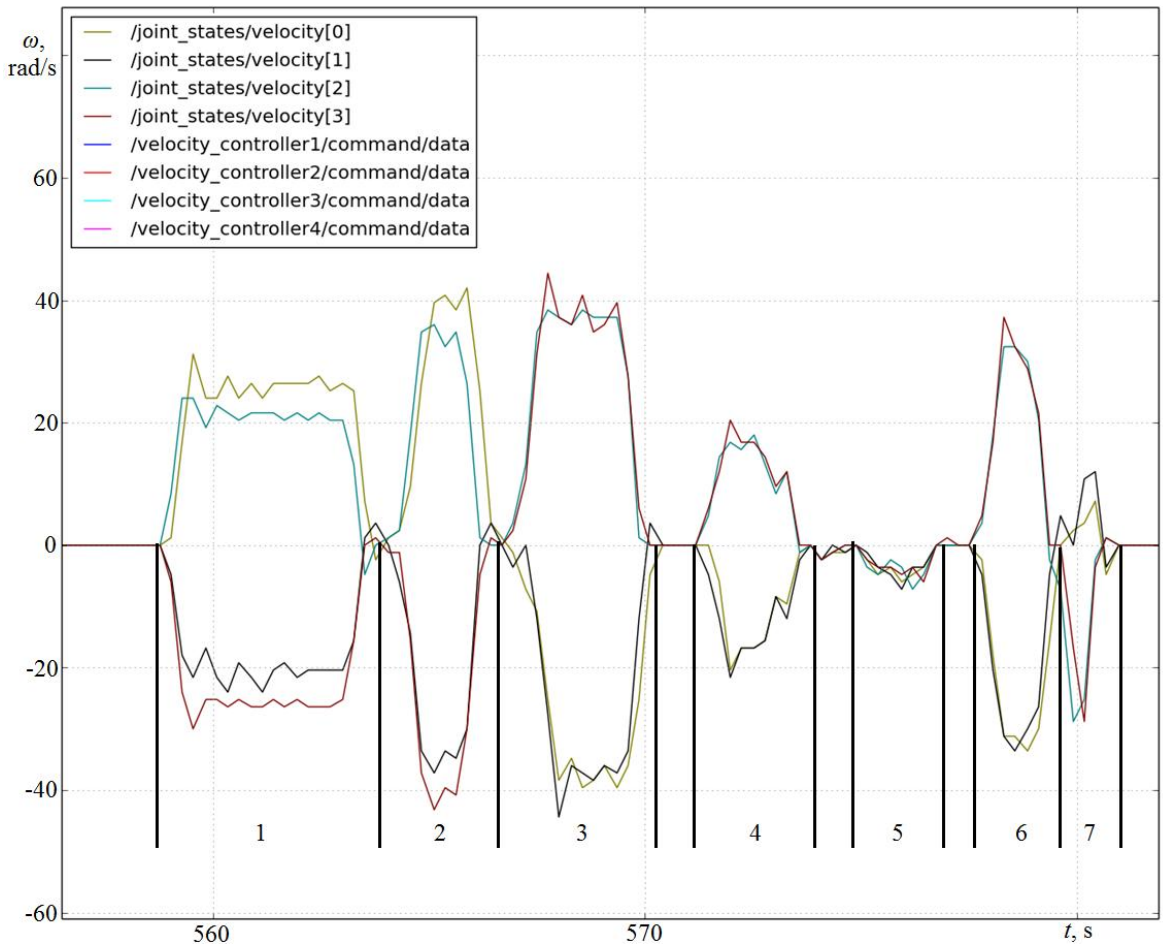


Figure 4.5. Velocity of each wheel displayed via *rqt_plot* tool while the robot is moving: 1 – left, 2 – left with greater speed, 3 – forward, 4 – forward with lower speed, 5 – left twist, 6 – forward, 7 – backward.

Due to the fact that each time the robot station polls and detects the controllers in random order, their numbering in the code is arbitrary. This can be seen in Figure 3.19, where */joint_state/velocity[0]* represents the 2nd motor, */joint_state/velocity[1]* – the 3rd motor, */joint_state/velocity[2]* – the 1st motor and */joint_state/velocity[3]* – the 4th motor. This drawback of the code is not significant and does not affect the functions of the code. However, it can be fixed for convenience by the next generation of researchers.

Nevertheless, as can be seen, the wheels of the robot follow the kinematics patterns described in Table 2. The signals from the registered velocity joint controller of each wheel and velocity commands sent to each wheel through the Xbox controller are displayed. Sharp peaks of velocities in the chart are due to encoder resolution and can be traced from the Maxon documentation (Maxon 2017).

5 SUMMARY AND CONCLUSION

The main task of the present project was development and implementation of the traction control system of the TIERA mobile robot developed in Laboratory of Intelligent Machines at LUT. During the research the construction of the TIERA robot developed by the previous researchers have been studied. Based on it, the mathematical model of the inverse kinematics problem has been created specifically for the mobile robot platform equipped with four Mecanum wheels. It has been utilized as a basis for the developed kinematics calculation service which takes into account how the rotation speed of each individual wheel affects the motion of the entire robot. The requirements of ROS middleware to the design process of traction systems for mobile robots and to the EPOS2 controller hardware have been studied. By utilizing these tools and techniques the code has been developed and integrated into the robot hardware.

All the malfunctions associated with specific features of the utilized equipment, such as communication interfaces or digital output configuration, have been eliminated by adjusting the code contents. In particular, the names of the serial ports through which communication takes place between the Advantech computer and the EPOS controllers are called from the code manually, rather than selected from the proposed list of ports. Moreover, the digital output values are set through particular file which contains all the configuration parameters of the motors. In order to avoid similar errors in the future, special logs have been introduced into the code.

The developed robot traction system provides it with the ability to move in any direction, including forward and sideways movement or turn on the spot. SSH-tunnel, embedded into the communication system, allows one to control the robot at safe distance from the operator who which sends commands remotely. The subsystems, developed by the other research members to control different parts of the robot and integrated into the unified system, allow using TIERA for various industrial, rescue and other tasks. The ground tests conducted in the Laboratory of Intelligent Machines have confirmed that:

1. Applied mathematical model of inverse kinematics adequately reflects the behavior of a real robotic platform equipped with four Mecanum wheels.

2. Designed traction system of the TIERA robot allows it to move freely in confined spaces.

3. Developed software provides all the required tools for the remote control of the robot.

The advantage of the developed traction system lies in its relative constructive simplicity, since it does not have a steering system and all movements to the side and rotations are performed due to the difference in the rotation speed of the wheels. Furthermore, it provides more maneuverability than the conventional steering traction systems and if necessary, it allows the operator to perform very precise movements without much effort. What is more, due to implementation of ROS the user codes can be written and compiled into one unified system even if the developers participating in the project use different programming languages.

However, the system also possesses some drawbacks which have to be taken into account. For example, efficiency of system operation strongly depends on reliability of the first controller (master drive) which transmits the data further to the slaves. Thus, if one controller of the serial connection fails the whole system ceases to work. Moreover, due to the limited data transfer speed of the CANopen protocol, the system is not capable of handling very complex multitasking instructions.

In general, the results obtained can be used as an example when developing the traction system of other mobile robots, taking into account their specific tasks and constructional design. For instance, the mathematical model of the inverse kinematics can be utilized for the robotic platform with the similar constructional design. The developed methods for solving specific problems (serial protocols issues, digital output configuration, et cetera) are also applicable for all robots using identical hardware.

The present traction system can be improved by developing the following methods. Foremost, when developing robot, one can exclude the ROS middleware from the system and use other software development tools such as LabVIEW, et cetera. The advantage of

this method is that all robot systems will function in a unified environment and the hardware system will not be overloaded with special tools for communication between various middleware tools.

Furthermore, the constructional design of the robot can be revised towards creating a robot that could safely navigate through rough terrain as the developed TIERA robot equipped with Mecanum wheels can experience difficulties in traction outdoors. For example, its suspension system can be modified in order to guarantee better stiffness and damping characteristics which should provide stability and reliability of the robotic arm operation on the run.

Besides, the software which will be able adjust the PID coefficients of each motor depending on the surface can be added upon improvement of the TIERA suspension. The surface mode should be chosen by the operator judging by the information he/she would get from the other TIERA systems in order to avoid surface recognition error.

LIST OF REFERENCES

Asama, H., Sato, M., Bogoni, L., Kaetsu, H., Mitsumoto A. & Endo I. 1995. Development of an omni-directional mobile robot with 3 DOF decoupling drive mechanism. Proceedings of 1995 IEEE International Conference on Robotics and Automation, Nagoya. Pp. 1925-1930.

BBC. 2013. Why Japan's 'Fukushima 50' remain unknown by Rupert Wingfield-Hayes. [BBC webpage]. [Referred 02.04.2018]. Available: http://www.bbc.com/news/world-asia-20707753.

Belzunce, A. 2015 Development of a Control System for a Tele Operated Mobile Robot [web document]. Lappeenranta: November 2015 [Referred 23.03.2018]. Master's thesis. Lappeenranta University of Technology, School of Energy Systems. 54 p. + appendixes 2 p. Available in PDF-file: https://www.doria.fi/bitstream/handle/10024/117725/Andres.Belzunce.pdf.

Ben-Tzvi, P., Goldenberg, A.A. & Zu, J.W. 2008. Design, simulations and optimization of a tracked mobile robot manipulator with hybrid locomotion and manipulation capabilities. Proceedings of the IEEE International Conference on Robotics and Automation. Pp. 2307-2312.

Chen, Y. & Dong, F. 2013. Robot machining: Recent development and future research issues. International Journal of Advanced Manufacturing Technology, 66:9-12. Pp. 1489-1497

Cheong, A., Lau, M.W.S., Foo, E. Hedley, J. & Ju Wen Bo 2016. Development of a Robotic Waiter System. 7th International Federation of Automatic Control (IFAC) Symposium on Mechatronic Systems (Mechatronics 2016): Leicestershire, UK, 2016. Pp. 681-686.

Doroftei, I., Grosu, V. & Spinu, V. 2008. Design and control of an omni-directional mobile

robot. Novel Algorithms and Techniques in Telecommunications, Automation and Industrial Electronics. Pp. 105-110.

Gfrerrer, A. 2008. Geometry and kinematics of the Mecanum wheel. Computer Aided Geometric Design, 25:9. Pp. 784-791.

Gopalakrishnan, B., Tirunellayi, S. & Todkar, R. 2004. Design and development of an autonomous mobile smart vehicle: a mechatronics application. Mechatronics, 14:5. Pp. 491-514.

GitHub. 2015. epos_hardware. [GitHub webpage]. [Referred 31.03.2018]. Available: https://github.com/RIVeR-Lab/epos_hardware.

Jia, Y., Song, X. & Xu, S.S.-D. 2013. Modeling and motion analysis of four-Mecanum wheel omni-directional mobile platform. 2013 CACS International Automatic Control Conference, CACS 2013 - Conference Digest. Pp. 328-333.

Kalpakjiaan, S. & Schmid, S.R. 2012. Manufacturing Engineering and Technology, 6th ed. USA: Prentice Hall. 1197 p.

Le, W. 2017. Design of traction transmission and suspension systems for an omni-directional mobile robot [web document]. Lappeenranta: November 2015 [Referred 17.04.2018]. Master's thesis. Lappeenranta University of Technology, School of Energy Systems. 58 p. + appendixes 3 p. Available in PDF-file: https://www.doria.fi/bitstream/handle/10024/117851/WeitingThesis.pdf.

Lin, L. & Shih, H. 2013. Modeling and Adaptive Control of an Omni-Mecanum-Wheeled Robot. Intelligent Control and Automation, 4. Pp. 166-179.

Maxon. 2012. An introduction to brushless DC motors. [Maxon motor web document]. [Referred 27.03.2018]. 24p. Available in PDF: https://www.maxonmotor.com/medias/sys_master/root/8803451338782/maxonECmotor-Notes.pdf.

Maxon. 2016a. EPOS2 Positioning Controllers. Getting Started. Edition: May 2016. [Maxon motor web document]. [Referred 30.03.2018]. 36p. Available in PDF: https://www.maxonmotor.nl/medias/sys_master/root/8821326774302/375711-Getting-Started-En.pdf.

Maxon. 2016b. EPOS2 Positioning Controllers. Feature Chart. Edition: April 2016. [Maxon motor web document]. [Referred 30.03.2018]. 9p. Available in PDF: https://www.maxonmotor.nl/medias/sys_master/root/8821706489886/EPOS2-Feature-Comparison-Chart-En-june-2016.pdf.

Maxon. 2016c. EPOS2 Positioning Controllers. Communication Guide. Edition: May 2016. [Maxon motor web document]. [Referred 30.03.2018]. 54p. Available in PDF: https://www.maxonmotor.com/medias/sys_master/root/8821325856798/EPOS2-Communication-Guide-En.pdf.

Maxon. 2016d. EPOS2 Positioning Controllers. Summary. Edition: April 2016. [Maxon motor web document]. [Referred 30.03.2018]. 4p. Available in PDF: http://www.ensatek.com.tr/image/urun/epos/1EPOS224-2.pdf.

Maxon. 2016e. EPOS Positioning Controllers. Command Library. Edition: November 2016. [Maxon motor web document]. [Referred 07.04.2018]. 172p. Available in PDF: https://www.maxonmotor.com/medias/sys_master/root/8823917281310/EPOS-Command-Library-En.pdf.

Maxon. 2017. EPOS2 Positioning Controllers. Firmware Specification. Edition: November 2017. [Maxon motor web document]. [Referred 04.04.2018]. 248p. Available in PDF: https://www.maxonmotor.com/medias/sys_master/root/8828099133470/EPOS2-Firmware-Specification-En.pdf.

Maxon. 2018a. Product overview. EC 60 Ø60 mm, brushless, 400 Watt, with Hall sensors. [Maxon motor webpage]. [Referred 27.03.2018]. Available: https://www.maxonmotor.com/maxon/view/product/167132.

Maxon. 2018b. EPOS. [Maxon motor webpage]. [Referred 30.03.2018]. Available: https://www.maxonmotor.com/maxon/view/content/EPOS-Detailsite.

Magyar, G., Sincak, P. & Krizsan, Z. 2015. Comparison Study of Robotic Middleware for Robotic Applications. Advances in Intelligent Systems and Computing, 316. Pp. 121-128.

Menshova, E. 2017. Development of LabVIEW platform for lower level subsystems of ROS-operated Mobile Assembly Robot [web document]. Lappeenranta: March 2017 [Referred 21.04.2018]. Master's thesis. Lappeenranta University of Technology, School of Energy Systems. 65 p. Available in PDF-file: https://www.doria.fi/bitstream/handle/10024/135176/masterthesis_menshova_ekaterina.pdf.

Mourioux, G., Novales, C., Poisson, G. & Vieyres, P. 2006. Omni-directional robot with spherical orthogonal wheels: Concepts and analyses. Proceedings of the IEEE International Conference on Robotics and Automation. Pp. 3374-3379

Muir, P.F. & Neuman, C.P. 1987. Kinematic modeling of wheeled mobile robots. Journal of Robotic Systems, 4:2. Pp. 281-340.

Poberezkin, E. 2017. Design and development of communication system for mobile robot [web document]. Lappeenranta: May 2017 [Referred 19.03.2018]. Master's thesis. Lappeenranta University of Technology, School of Energy Systems. 72 p. + appendixes 10 p. Available in PDF-file: http://www.doria.fi/bitstream/handle/10024/135060/mastersthesis_poberezkin_efim.pdf.

Robustel. 2018. R3000 Router [Robustel webpage]. [Referred 30.03.2018]. Available: http://www.robustel.com/products/gorugged-industrial-cellular-r/r3000-router.html.

ROS. 2012a. Nodes. [ROS Wiki webpage]. Updated February 3, 2012. [Referred 23.03.2018]. Available: http://wiki.ros.org/ROS/Nodes.

ROS. 2012b. Services. [ROS Wiki webpage]. Updated February 3, 2012. [Referred 25.03.2018]. Available: http://wiki.ros.org/ROS/Services.

ROS. 2012c. Master. [ROS Wiki webpage]. Updated January 15, 2012. [Referred 25.03.2018]. Available: http://wiki.ros.org/ROS/Master.

ROS. 2013. Parameter Server. [ROS Wiki webpage]. Updated August 7, 2013. [Referred 25.03.2018]. Available: http://wiki.ros.org/Parameter%20Server.

ROS. 2014a. Introduction. [ROS Wiki webpage]. Updated May 22, 2014. [Referred 20.03.2018]. Available: http://wiki.ros.org/ROS/Introduction.

ROS. 2014b. Concepts. [ROS Wiki webpage]. Updated June 21, 2014. [Referred 23.03.2018]. Available: http://wiki.ros.org/ROS/Concepts.

ROS. 2014c. ROS. Technical Overview. [ROS Wiki webpage]. Updated June 15, 2014. [Referred 28.04.2018]. Available: http://wiki.ros.org/ROS/Technical%20Overview.

ROS. 2014d. Topics. [ROS Wiki webpage]. Updated June 1, 2014. [Referred 24.03.2018]. Available: http://wiki.ros.org/ROS/Topics.

ROS. 2015. epos_hardware. Package Summary [ROS Wiki webpage]. Updated March 26, 2015. [Referred 29.04.2018]. Available: http://wiki.ros.org/epos_hardware.

ROS. 2016a. Ubuntu install of ROS Indigo. [ROS Wiki webpage]. Updated October 28, 2016. [Referred 20.03.2018]. Available: http://wiki.ros.org/indigo/Installation/Ubuntu.

ROS. 2016b. Messages. [ROS Wiki webpage]. Updated August 26, 2016. [Referred 24.03.2018]. Available: http://wiki.ros.org/ROS/Messages.

ROS. 2016c. roscore. [ROS Wiki webpage]. Updated November 25, 2016. [Referred 25.03.2018]. Available: http://wiki.ros.org/ROS/roscore.

ROS. 2016d. epos_hardware::EposHardware Class Reference. [ROS Wiki webpage]. Updated February 12, 2016. [Referred 01.04.2018]. Available: http://docs.ros.org/

indigo/api/epos_hardware/html/classepos__hardware_1_1EposHardware.html.
ROS. 2017. Documentation. [ROS Wiki webpage]. Updated March 31, 2017. [Referred 20.03.2018]. Available: http://wiki.ros.org.

ROS. 2018a. Master. [ROS Wiki webpage]. Updated January 15, 2018. [Referred 31.03.2018]. Available:http://wiki.ros.org/Master.

ROS. 2018a. roslaunch. [ROS Wiki webpage]. Updated February 10, 2018. [Referred 31.03.2018]. Available: http:// http://wiki.ros.org/roslaunch.

Saha, S.K., Angeles, J. & Darcovich, J. 1995. The design of kinematically isotropic rolling robots with omnidirectional wheels. Mechanism and Machine Theory, 30:8. Pp. 1127-1137.

Scopus. 2018. Analyze search results [web database]. [Referred 25.04.2018]. Available: https://www.scopus.com/term/analyzer.uri?sid=29dc56e9119e738c7c5f98803e8b0b5d&origin=resultslist&src=s&sort=cp-f&sdt=cl&sot=b&sessionSearchId=29dc56e9119e738c7c5f98803e8b0b5d&count=3855&analyzeResults=Analyze.

Song, J.B. & Byun, K.S. 2004. Design and control of a four-wheeled omnidirectional mobile robot with steerable omnidirectional wheels. Journal of Robotic Systems, 21:4. Pp. 193-208.

Soroka, I. 2016. Design and implementation of machine vision system for the Mobile Assembly Robot [web document]. Lappeenranta: February 2016 [Referred 21.04.2018]. Master's thesis. Lappeenranta University of Technology, School of Energy Systems. 60 p. + appendixes 1 p. Available in PDF-file: http://www.doria.fi/bitstream/handle/10024/123373/IGOR%20SOROKA.pdf.

SSH. 2017. SSH command. Command line usage. [SSH webpage]. Updated October 10, 2017. [Referred 07.04.2018]. Available: https://www.ssh.com/ssh/command/.

Tlale, N. & de Villiers, M. 2008. Kinematics and Dynamics Modelling of a Mecanum

Wheeled Mobile Platform. 15th International Conference on Mechatronics and Machine Vision in Practice (M2VIP'08), Auckland, New Zealand, 2008. Pp. 657-662.

Viboonchaicheep, P., Shimada, A. & Kosaka, Y. 2003. Position Rectification Control for Mecanum Wheeled Omni-directional Vehicles. IECON Proceedings (Industrial Electronics Conference), 1. Pp. 854-859.

Wakchaure, K.N., Bhaskar, S.V., Thakur, A.G. & Modak, G.S. 2011. Kinematics modelling of Mechanum wheeled mobile platform. Intelligent Control and Automation, 2. Pp. 155-159.

Walker Industrial. 2014. ARK-3440 A2. Specifications [Walker Industrial web document]. [Referred 30.03.2018]. Edition: January 2014. 2p. Available in PDF: http://www.walkerindustrial.com/v/vspfiles/pdf/datasheet/Advantech/ARK-3440%20A2_DS(01.15.14)20140122150106.pdf.

Walker Industrial. 2018. ARK-3440F-U5A2E – Advantech barebones unconfigured PC [Walker Industrial webpage]. [Referred 30.03.2018]. Available: http://www.walkerindustrial.com/ARK-3440F-U5A2E-Advantech-p/ark-3440f-u5a2e.htm.

Xbox. 2018. Xbox 360 wired and wireless controllers [Xbox webpage]. [Referred 10.04.2018]. Available: https://support.xbox.com/en-US/xbox-360/accessories/controllers.

APPENDIX I

Maxon EC60 400W motor characteristics (Maxon 2018a).

Values at nominal voltage	
Nominal voltage	48 V
No load speed	5370 rpm
No load current	670 mA
Nominal speed	4960 rpm
Nominal torque (max. continuous torque)	768 mNm
Nominal current (max. continuous current)	9.56 A
Stall torque	11800 mNm
Stall current	139 A
Max. efficiency	87 %
Mechanical data	
Bearing type	ball bearings
Max. speed	7000 rpm
Max. axial load (dynamic)	24 N
Max. force for press fits (static)	390 N
(static, shaft supported)	6000 N
Max. radial load	240 N, 5 mm from flange
Other specifications	
Number of pole pairs	1
Number of phases	3
Number of autoclave cycles	0

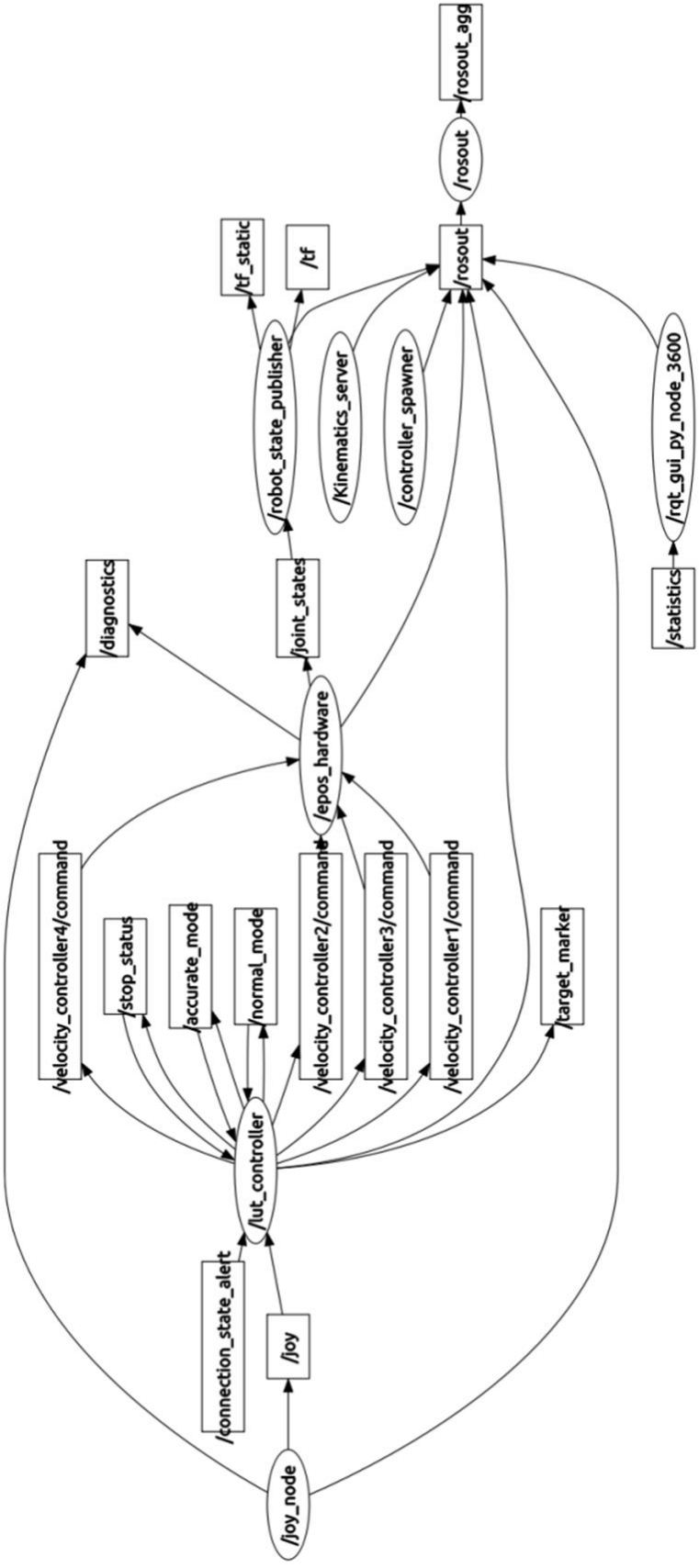
APPENDIX II

Technical characteristics of Advantech ARK-3440F-U5A2E (Walker Industrial 2014).

Processor System	CPU	Intel Core i7 610E (Dual Core)
	Frequency	2.53 GHz
	BIOS	AMI 16 Mbit SPI BIOS
Memory	Max. Capacity	8 GB
IO interface	Serial Interface	2 x RS232, 1 x RS232/422/485 (w/auto flow control)
	USB Interface	6 x USB ports, compliant with USB 2.0
	Parallel Port	Supports D-sub 25-pin connector (optional)
	Digital I/O	-
Mechanical	Construction	Aluminum housing
	Mounting	Desk/wall-mounting
	Dimensions (W x H x D)	220 mm x 117 mm x 200 mm (8.66" x 4.61" x 7.87")

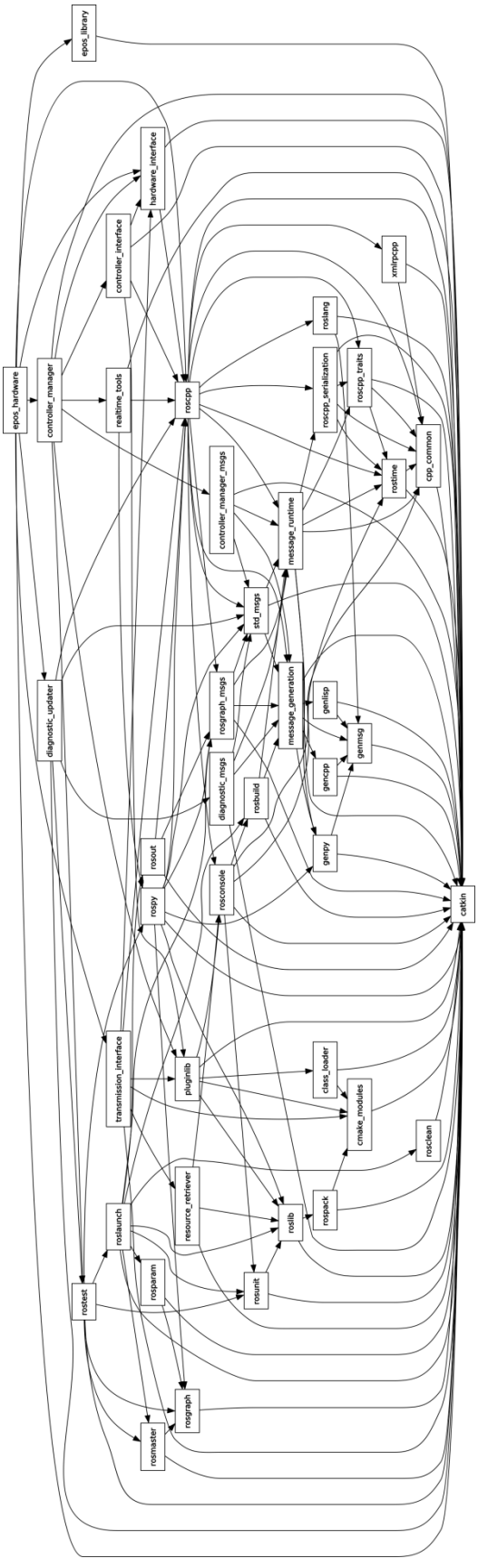
APPENDIX III

Dependencies of *epos_hardware* and *lut_track* packages obtained by *rqt_plot* tool.



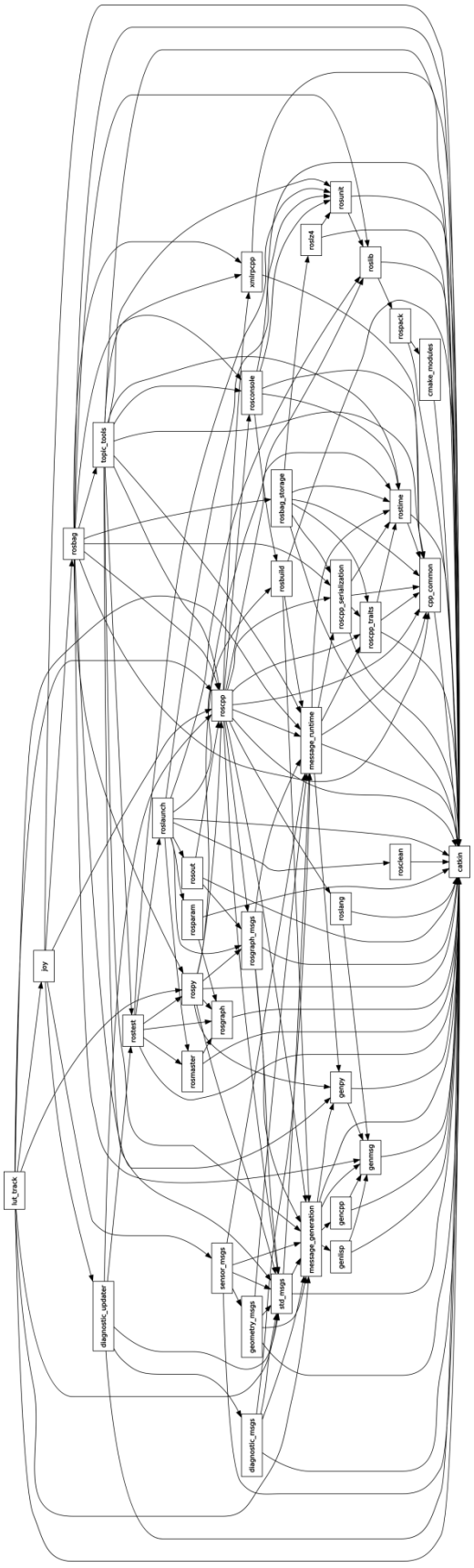
APPENDIX IV

Diagram of the *epos_hardware* package dependencies.



APPENDIX V

Diagram of the *lut_track* package dependencies.



Code of *utils.cpp* file for initializing the connection between the Advantech PC and EPOS2 70/10 digital controller.

```
#include "epos_hardware/utils.h"
#include <boost/foreach.hpp>
#include <sstream>

#define MAX_STRING_SIZE 1000

bool SerialNumberFromHex(const std::string& str, uint64_t* serial_number)
{
  std::stringstream ss;
  ss << std::hex << str;
  ss >> *serial_number;
  return true;
}

int GetErrorInfo(unsigned int error_code, std::string* error_string) {
  char buffer[MAX_STRING_SIZE];
  int result;
  if(result = VCS_GetErrorInfo(error_code, buffer, MAX_STRING_SIZE)) {
    *error_string = buffer;
  }
  return result;
}

int  GetDeviceNameList(std::vector<std::string>*  device_names,  unsigned
int* error_code) {
  char buffer[MAX_STRING_SIZE];
  int end_of_selection;
  int result;

  result  =  VCS_GetDeviceNameSelection  (true,  buffer,  MAX_STRING_SIZE,
&end_of_selection, error_code);
  if(!result)
    return result;
  device_names->push_back(buffer);

  while(!end_of_selection) {
    result = VCS_GetDeviceNameSelection (false, buffer, MAX_STRING_SIZE,
&end_of_selection, error_code);
    if(!result)
      return result;
    device_names->push_back(buffer);
  }
  return 1;
}

int      GetProtocolStackNameList(const      std::string      device_name,
std::vector<std::string>* protocol_stack_names, unsigned int* error_code)
{
  char buffer[MAX_STRING_SIZE];
  int end_of_selection; //BOOL
  int result;

  result = VCS_GetProtocolStackNameSelection ((char*)device_name.c_str(),
```

```
true, buffer, MAX_STRING_SIZE, &end_of_selection, error_code);
  if(!result)
    return result;
  protocol_stack_names->push_back(buffer);

  while(!end_of_selection) {
    result              =              VCS_GetProtocolStackNameSelection
((char*)device_name.c_str(),     false,     buffer,     MAX_STRING_SIZE,
&end_of_selection, error_code);
    if(!result)
      return result;
    protocol_stack_names->push_back(buffer);
  }
  return 1;
}

int   GetInterfaceNameList   (const   std::string   device_name,   const
std::string        protocol_stack_name,        std::vector<std::string>*
interface_names, unsigned int* error_code) {
  char buffer[MAX_STRING_SIZE];
  int end_of_selection; //BOOL
  int result;

  result  =  VCS_GetInterfaceNameSelection  ((char*)device_name.c_str(),
(char*)protocol_stack_name.c_str(),   true,   buffer,   MAX_STRING_SIZE,
&end_of_selection, error_code);
  if(!result)
    return result;
  interface_names->push_back(buffer);

  while(!end_of_selection) {
    result  =  VCS_GetInterfaceNameSelection  ((char*)device_name.c_str(),
(char*)protocol_stack_name.c_str(),   false,   buffer,   MAX_STRING_SIZE,
&end_of_selection, error_code);
    if(!result)
      return result;
    interface_names->push_back(buffer);
  }
  return 1;
}

int  GetPortNameList(const  std::string  device_name,  const  std::string
protocol_stack_name,       const       std::string       interface_name,
std::vector<std::string>* port_names, unsigned int* error_code) {
  char buffer[MAX_STRING_SIZE];
  int end_of_selection; //BOOL
  int result;

  result    =    VCS_GetPortNameSelection    ((char*)device_name.c_str(),
(char*)protocol_stack_name.c_str(),  (char*)interface_name.c_str(),  true,
buffer, MAX_STRING_SIZE, &end_of_selection, error_code);
  if(!result)
    return result;
  port_names->push_back(buffer);

  while(!end_of_selection) {
           result = VCS_GetPortNameSelection ((char*)device_name.c_str(),
```

```
(char*)protocol_stack_name.c_str(), (char*)interface_name.c_str(), false,
buffer, MAX_STRING_SIZE, &end_of_selection, error_code);
    if(!result)
      return result;
    port_names->push_back(buffer);
  }
  return 1;
}

int GetBaudrateList(const std::string device_name, const std::string
protocol_stack_name, const std::string interface_name, const std::string
port_name, std::vector<unsigned int>* baudrates, unsigned int*
error_code) {
  unsigned int baudrate;
  int end_of_selection; //BOOL
  int result;

  result = VCS_GetBaudrateSelection ((char*)device_name.c_str(),
(char*)protocol_stack_name.c_str(), (char*)interface_name.c_str(),
(char*)port_name.c_str(), true, &baudrate, &end_of_selection,
error_code);
  if(!result)
    return result;
  baudrates->push_back(baudrate);

  while(!end_of_selection) {

    result = VCS_GetBaudrateSelection ((char*)device_name.c_str(),
(char*)protocol_stack_name.c_str(), (char*)interface_name.c_str(),
(char*)port_name.c_str(), false, &baudrate, &end_of_selection,
error_code);
    if(!result)
      return result;
    baudrates->push_back(baudrate);
  }
  return 1;
}

EposFactory::EposFactory()
DeviceHandlePtr EposFactory::CreateDeviceHandle(const std::string
device_name, const std::string protocol_stack_name, const std::string
interface_name, const std::string port_name, unsigned int*
error_code) {
  const std::string key = device_name + '/' + protocol_stack_name + '/' +
interface_name + '/' + port_name;
  DeviceHandlePtr handle;
  if(!(handle = existing_handles[key].lock())) { // Handle exists
    void* raw_handle = VCS_OpenDevice ((char*)device_name.c_str(),
(char*)protocol_stack_name.c_str(), (char*)interface_name.c_str(),
(char*)port_name.c_str(), error_code);
    if(!raw_handle) // failed to open device
      return DeviceHandlePtr();

    handle = DeviceHandlePtr(new DeviceHandle(raw_handle));
    existing_handles[key] = handle;
  }
  return handle;
```

```
}

NodeHandlePtr       EposFactory::CreateNodeHandle(const       std::string
device_name,  const  std::string  protocol_stack_name,  const  std::string
interface_name, const uint64_t serial_number, unsigned int* error_code) {
  std::vector<EnumeratedNode> nodes;

  EnumerateNodes(device_name,    protocol_stack_name,    interface_name,
&nodes, error_code);
  BOOST_FOREACH(const EnumeratedNode& node, nodes) {
    if(node.serial_number == serial_number) {
      return CreateNodeHandle(node, error_code);
    }
  }
  return NodeHandlePtr();
}

NodeHandlePtr  EposFactory::CreateNodeHandle(const  EnumeratedNode&  node,
unsigned int* error_code) {
  DeviceHandlePtr  device_handle  =  CreateDeviceHandle  (node.device_name,
node.protocol_stack_name,      node.interface_name,      node.port_name,
error_code);
  if(!device_handle)
    return NodeHandlePtr();
  return NodeHandlePtr(new NodeHandle(device_handle, node.node_id));
}

int  EposFactory::EnumerateNodes(const  std::string  device_name,  const
std::string protocol_stack_name, const std::string interface_name, const
std::string port_name, std::vector<EnumeratedNode>* nodes, unsigned int*
error_code) {
  DeviceHandlePtr handle;

//  std::cout << "\tERROR LOG port in :" << port_name << std::endl;

    if(!(handle  =  CreateDeviceHandle(device_name,  protocol_stack_name,
interface_name, port_name, error_code))){
    std::cout  <<  "CreateDeviceHandle  failed  with  error_code="  <<
*error_code << std::endl;

    return 0;
  }
  for(unsigned short i = 1; i < 127; ++i) {
    EnumeratedNode node;
    node.device_name = device_name;
    node.protocol_stack_name = protocol_stack_name;
    node.interface_name = interface_name;
    node.port_name = port_name;
    node.node_id = i;

    //    std::cout << "Try GetVersion..." << std::endl;
    if(!VCS_GetVersion(handle->ptr,      i,      &node.hardware_version,
&node.software_version,                      &node.application_number,
&node.application_version, error_code)){
      return 1;
    }
    unsigned int bytes_read;
```

```
   //    std::cout << "Try GetObject..." << std::endl;
    if(!VCS_GetObject(handle->ptr, i, 0x2004, 0x00, &node.serial_number,
8, &bytes_read, error_code)){
      node.serial_number = 0;
    }
    nodes->push_back(node);
  }
  return 1;
}

int EposFactory::EnumerateNodes(const std::string device_name, const
std::string protocol_stack_name, const std::string interface_name,
std::vector<EnumeratedNode>* nodes, unsigned int* error_code) {
  std::vector<std::string> port_names;
  GetPortNameList(device_name,    protocol_stack_name,    interface_name,
&port_names, error_code);
  std::cout << "Number of ports available: " << port_names.size() <<
std::endl;
  port_names.clear();
  port_names.push_back("/dev/ttyS0");
  port_names.push_back("/dev/ttyS1");
  if(true) {

   BOOST_FOREACH(const std::string& port_name, port_names) {
      std::cout << "ERROR_LOG: Try init device_name:" << device_name <<
std::endl;
      std::cout   <<   "ERROR_LOG:   Try   init   protocol_stack:"   <<
protocol_stack_name << std::endl;
      std::cout << "ERROR_LOG: Try init interface:" << interface_name <<
std::endl;
      std::cout << "ERROR_LOG: Try init port:" << port_name << std::endl;
      if(!EnumerateNodes(device_name,                protocol_stack_name,
interface_name, port_name, nodes, error_code)){
           return 0;
      }
    }
    return 1;
  }
  else
    return 0;
}
```

Code of *epos.cpp* file for configuring Maxon motor parameters in EPOS2 70/10 digital controllers.

```
#include "epos_hardware/epos.h"
#include <boost/foreach.hpp>

namespace epos_hardware {

Epos::Epos(const std::string& name,
              ros::NodeHandle& nh, ros::NodeHandle& config_nh,
              EposFactory* epos_factory,
              hardware_interface::ActuatorStateInterface& asi,
              hardware_interface::VelocityActuatorInterface& avi,
              hardware_interface::PositionActuatorInterface& api)
  : name_(name), config_nh_(config_nh), diagnostic_updater_(nh,
config_nh), epos_factory_(epos_factory),
    has_init_(false),
    position_(0), velocity_(0), effort_(0), current_(0), statusword_(0),
    position_cmd_(0), velocity_cmd_(0) {

  valid_ = true;
  if(!config_nh_.getParam("actuator_name", actuator_name_)) {
    ROS_ERROR("You must specify an actuator name");
    valid_ = false;
  }

  std::string serial_number_str;
  if(!config_nh_.getParam("serial_number", serial_number_str)) {
    ROS_ERROR("You must specify a serial number");
    valid_ = false;
  }
  else {
    ROS_ASSERT(SerialNumberFromHex(serial_number_str, &serial_number_));
  }

  std::string operation_mode_str;
  if(!config_nh_.getParam("operation_mode", operation_mode_str)) {
    ROS_ERROR("You must specify an operation mode");
    valid_ = false;
  }
  else {
    if(operation_mode_str == "profile_position") {
      operation_mode_ = PROFILE_POSITION_MODE;
    }
    else if(operation_mode_str == "profile_velocity") {
      operation_mode_ = PROFILE_VELOCITY_MODE;
    }
    else {
      ROS_ERROR_STREAM(operation_mode_str << " is not a valid operation
mode");
      valid_ = false;
    }
  }

  ROS_INFO_STREAM(actuator_name_);
     hardware_interface::ActuatorStateHandle state_handle(actuator_name_,
```

```
&position_, &velocity_, &effort_);
  asi.registerHandle(state_handle);

  hardware_interface::ActuatorHandle position_handle(state_handle,
&position_cmd_);
  api.registerHandle(position_handle);
  hardware_interface::ActuatorHandle velocity_handle(state_handle,
&velocity_cmd_);
  avi.registerHandle(velocity_handle);

  diagnostic_updater_.setHardwareID(serial_number_str);
  std::stringstream motor_diagnostic_name_ss;
  motor_diagnostic_name_ss << name << ": " << "Motor";
  diagnostic_updater_.add(motor_diagnostic_name_ss.str(),
boost::bind(&Epos::buildMotorStatus, this, _1));
  std::stringstream motor_output_diagnostic_name_ss;
  motor_output_diagnostic_name_ss << name << ": " << "Motor Output";
  diagnostic_updater_.add(motor_output_diagnostic_name_ss.str(),
boost::bind(&Epos::buildMotorOutputStatus, this, _1));
}

Epos::~Epos() {
  unsigned int error_code;
  if(node_handle_)
    VCS_SetDisableState(node_handle_->device_handle->ptr, node_handle_-
>node_id, &error_code);
}

class ParameterSetLoader {
public:
  ParameterSetLoader(ros::NodeHandle nh) : nh_(nh){}
  ParameterSetLoader(ros::NodeHandle parent_nh, const std::string& name)
: nh_(parent_nh, name){}
  template <class T> ParameterSetLoader& param(const std::string& name,
T& value) {
    if(nh_.getParam(name, value))
      found_.push_back(name);
    else
      not_found_.push_back(name);
    return *this;
  }
  bool all_or_none(bool& found_all) {
    if(not_found_.size() == 0) {
      found_all = true;
      return true;
    }
    if(found_.size() == 0) {
      found_all = false;
      return true;
    }

    ROS_ERROR_STREAM("Expected all or none parameter set: (" <<
nh_.getNamespace() << ")");
    BOOST_FOREACH(const std::string& name, found_) {
      ROS_ERROR_STREAM("\tFound: " << nh_.resolveName(name));
    }
    BOOST_FOREACH(const std::string& name, not_found_) {
```

```
      ROS_ERROR_STREAM("\tExpected: " << nh_.resolveName(name));
    }
    return false;
  }

private:
  ros::NodeHandle nh_;
  std::vector<std::string> found_;
  std::vector<std::string> not_found_;
};

#define VCS(func, ...)
  if(!VCS_##func(node_handle_->device_handle->ptr, node_handle_->node_id,
__VA_ARGS__, &error_code)) {
    ROS_ERROR("Failed to "#func);
    return false;
  }

#define VCS_FROM_SINGLE_PARAM_REQUIRED(nh, type, name, func)
  type name;
  if(!nh.getParam(#name, name)) {
    ROS_ERROR_STREAM(nh.resolveName(#name) << " not specified");
    return false;
  }
  else {
    VCS(func, name);
  }
#define VCS_FROM_SINGLE_PARAM_OPTIONAL(nh, type, name, func)
  bool name##_set;
  type name;

  if(name##_set = nh.getParam(#name, name)) {

    VCS(func, name);
  }


bool Epos::init() {
  if(!valid_) {
    ROS_ERROR_STREAM("Not Initializing: 0x" << std::hex << serial_number_
<< ", initial construction failed");
    return false;
  }

  ROS_INFO_STREAM("Initializing: 0x" << std::hex << serial_number_);
  unsigned int error_code;
  node_handle_ = epos_factory_->CreateNodeHandle("EPOS2", /*"MAXON SERIAL
V2"*/"MAXON_RS232", /*"USB"*/"RS232", serial_number_, &error_code);
  if(!node_handle_) {
    ROS_ERROR("Could not find motor");
    return false;
  }
  ROS_INFO_STREAM("Found Motor");

  if(!VCS_SetProtocolStackSettings(node_handle_->device_handle->ptr,
1000000, 500, &error_code)) {
    ROS_ERROR("Failed to SetProtocolStackSettings");
```

```
    return false;
  }
  ROS_INFO("BAUDRATE: 1000000, TIMEOUT 500 Succeeded"); ////////////
ADDED

  if(!VCS_SetDisableState(node_handle_->device_handle->ptr, node_handle_-
>node_id, &error_code)) {
    ROS_ERROR("Failed to SetDisableState");
    return false;
  }

  VCS(SetOperationMode, operation_mode_);
  //ROS_INFO_STREAM("OPERATION MODE TIMEOUT 500 Succeeded");////////////
ADDED
  std::string fault_reaction_str;
#define SET_FAULT_REACTION_OPTION(val)
  do {
    unsigned int length = 2;
    unsigned int bytes_written;
    int16_t data = val;
    VCS(SetObject, 0x605E, 0x00, &data, length, &bytes_written);
  } while(true)

  if(config_nh_.getParam("fault_reaction_option", fault_reaction_str)) {
    if(fault_reaction_str == "signal_only") {
      SET_FAULT_REACTION_OPTION(-1);
    }
    else if(fault_reaction_str == "disable_drive") {
      SET_FAULT_REACTION_OPTION(0);
    }
    else if(fault_reaction_str == "slow_down_ramp") {
      SET_FAULT_REACTION_OPTION(1);
    }
    else if(fault_reaction_str == "slow_down_quickstop") {
      SET_FAULT_REACTION_OPTION(2);
    }
    else {
      ROS_ERROR_STREAM(fault_reaction_str << " is not a valid fault
reaction option");
      return false;
    }
  }

  ROS_INFO("Configuring Motor");
  {
    nominal_current_ = 0;
    max_current_ = 0;
    ros::NodeHandle motor_nh(config_nh_, "motor");

    VCS_FROM_SINGLE_PARAM_REQUIRED(motor_nh, int, type, SetMotorType);

    {
      bool dc_motor;
      double nominal_current;
      double max_output_current;
      double thermal_time_constant;
      if(!ParameterSetLoader(motor_nh, "dc_motor")
```

```
            .param("nominal_current", nominal_current)
            .param("max_output_current", max_output_current)
            .param("thermal_time_constant", thermal_time_constant)
            .all_or_none(dc_motor))
          return false;
    if(dc_motor){
          nominal_current_ = nominal_current;
          max_current_ = max_output_current;
          VCS(SetDcMotorParameter,
              (int)(1000 * nominal_current), // A -> mA
              (int)(1000 * max_output_current), // A -> mA
              (int)(10 * thermal_time_constant) // s -> 100ms
              );
    }
  }


  {
    bool ec_motor;
    double nominal_current;
    double max_output_current;
    double thermal_time_constant;
    int number_of_pole_pairs;
    if(!ParameterSetLoader(motor_nh, "ec_motor")
          .param("nominal_current", nominal_current)
          .param("max_output_current", max_output_current)
          .param("thermal_time_constant", thermal_time_constant)
          .param("number_of_pole_pairs", number_of_pole_pairs)
          .all_or_none(ec_motor))
          return false;

    if(ec_motor) {
          nominal_current_ = nominal_current;
          max_current_ = max_output_current;
          VCS(SetEcMotorParameter,
              (int)(1000 * nominal_current), // A -> mA
              (int)(1000 * max_output_current), // A -> mA
              (int)(10 * thermal_time_constant), // s -> 100ms
              number_of_pole_pairs);
    }
  }
}

ROS_INFO("Configuring Sensor");
{
  ros::NodeHandle sensor_nh(config_nh_, "sensor");

  VCS_FROM_SINGLE_PARAM_REQUIRED(sensor_nh, int, type, SetSensorType);

  {
    bool incremental_encoder;
    int resolution;
    bool inverted_polarity;
    if(!ParameterSetLoader(sensor_nh, "incremental_encoder")
          .param("resolution", resolution)
          .param("inverted_polarity", inverted_polarity)
          .all_or_none(incremental_encoder))
```

```
            return false;
      if(incremental_encoder) {
            VCS(SetIncEncoderParameter, resolution, inverted_polarity);
      }
    }

    {
      bool hall_sensor;
      bool inverted_polarity;

      if(!ParameterSetLoader(sensor_nh, "hall_sensor")
            .param("inverted_polarity", inverted_polarity)
            .all_or_none(hall_sensor))
            return false;
      if(hall_sensor) {
            VCS(SetHallSensorParameter, inverted_polarity);
      }
    }

    {
      bool ssi_absolute_encoder;
      int data_rate;
      int number_of_multiturn_bits;
      int number_of_singleturn_bits;
      bool inverted_polarity;

      if(!ParameterSetLoader(sensor_nh, "ssi_absolute_encoder")
            .param("data_rate", data_rate)
            .param("number_of_multiturn_bits", number_of_multiturn_bits)
            .param("number_of_singleturn_bits",
number_of_singleturn_bits)
            .param("inverted_polarity", inverted_polarity)
            .all_or_none(ssi_absolute_encoder))
            return false;
      if(ssi_absolute_encoder) {
            VCS(SetSsiAbsEncoderParameter,
                data_rate,
                number_of_multiturn_bits,
                number_of_singleturn_bits,
                inverted_polarity);
      }
    }

  }

  {
    ROS_INFO("Configuring Safety");
    ros::NodeHandle safety_nh(config_nh_, "safety");

    VCS_FROM_SINGLE_PARAM_OPTIONAL(safety_nh, int, max_following_error,
SetMaxFollowingError);
    VCS_FROM_SINGLE_PARAM_OPTIONAL(safety_nh, int, max_profile_velocity,
SetMaxProfileVelocity);
    VCS_FROM_SINGLE_PARAM_OPTIONAL(safety_nh, int, max_acceleration,
SetMaxAcceleration);
    if(max_profile_velocity_set)
      max_profile_velocity_ = max_profile_velocity;
```

```
    else
      max_profile_velocity_ = -1;
  }

  {
    ROS_INFO("Configuring Position Regulator");
    ros::NodeHandle position_regulator_nh(config_nh_,
"position_regulator");
    {
      bool position_regulator_gain;
      int p, i, d;
      if(!ParameterSetLoader(position_regulator_nh, "gain")
            .param("p", p)
            .param("i", i)
            .param("d", d)
            .all_or_none(position_regulator_gain))
          return false;
      if(position_regulator_gain){
          VCS(SetPositionRegulatorGain, p, i, d);
      }
    }

    {
      bool position_regulator_feed_forward;
      int velocity, acceleration;
      if(!ParameterSetLoader(position_regulator_nh, "feed_forward")
            .param("velocity", velocity)
            .param("acceleration", acceleration)
            .all_or_none(position_regulator_feed_forward))
          return false;
      if(position_regulator_feed_forward){
          VCS(SetPositionRegulatorFeedForward, velocity, acceleration);
      }
    }
  }

  {
    ROS_INFO("Configuring Velocity Regulator");
    ros::NodeHandle velocity_regulator_nh(config_nh_,
"velocity_regulator");
    {
      bool velocity_regulator_gain;
      int p, i;
      if(!ParameterSetLoader(velocity_regulator_nh, "gain")
            .param("p", p)
            .param("i", i)
            .all_or_none(velocity_regulator_gain))
          return false;
      if(velocity_regulator_gain){
          VCS(SetVelocityRegulatorGain, p, i);
      }
    }

    {
      bool velocity_regulator_feed_forward;
      int velocity, acceleration;
      if(!ParameterSetLoader(velocity_regulator_nh, "feed_forward")
```

```
            .param("velocity", velocity)
            .param("acceleration", acceleration)
            .all_or_none(velocity_regulator_feed_forward))
          return false;
      if(velocity_regulator_feed_forward){
          VCS(SetVelocityRegulatorFeedForward, velocity, acceleration);
      }
    }
  }


  {
    ROS_INFO("Configuring Current Regulator");
    ros::NodeHandle current_regulator_nh(config_nh_,
"current_regulator");
    {
      bool current_regulator_gain;
      int p, i;
      if(!ParameterSetLoader(current_regulator_nh, "gain")
            .param("p", p)
            .param("i", i)
            .all_or_none(current_regulator_gain))
          return false;
      if(current_regulator_gain){
          VCS(SetCurrentRegulatorGain, p, i);
      }
    }
  }


  {
    ROS_INFO("Configuring Position Profile");
    ros::NodeHandle position_profile_nh(config_nh_, "position_profile");
    {
      bool position_profile;
      int velocity, acceleration, deceleration;
      if(!ParameterSetLoader(position_profile_nh)
            .param("velocity", velocity)
            .param("acceleration", acceleration)
            .param("deceleration", deceleration)
            .all_or_none(position_profile))
          return false;
      if(position_profile){
          VCS(SetPositionProfile, velocity, acceleration, deceleration);
      }
    }

    {
      bool position_profile_window;
      int window;
      double time;
      if(!ParameterSetLoader(position_profile_nh, "window")
            .param("window", window)
            .param("time", time)
            .all_or_none(position_profile_window))
          return false;
      if(position_profile_window){
```

```
            VCS(EnablePositionWindow,
                window,
                (int)(1000 * time) // s -> ms
                );
        }
      }
    }

    {
      ROS_INFO("Configuring Velocity Profile");
      ros::NodeHandle velocity_profile_nh(config_nh_, "velocity_profile");
      {
        bool velocity_profile;
        int acceleration, deceleration;
        if(!ParameterSetLoader(velocity_profile_nh)
              .param("acceleration", acceleration)
              .param("deceleration", deceleration)
              .all_or_none(velocity_profile))
            return false;
        if(velocity_profile){
            VCS(SetVelocityProfile, acceleration, deceleration);
        }
      }

      {
        bool velocity_profile_window;
        int window;
        double time;
        if(!ParameterSetLoader(velocity_profile_nh, "window")
              .param("window", window)
              .param("time", time)
              .all_or_none(velocity_profile_window))
            return false;
        if(velocity_profile_window){
            VCS(EnableVelocityWindow,
                window,
                (int)(1000 * time) // s -> ms
                );
        }
      }

    }

    {
      ROS_INFO("Configuring Digital Outputs aka Brakes");
      ros::NodeHandle outputs_nh(config_nh_, "outputs");

      VCS_FROM_SINGLE_PARAM_OPTIONAL(outputs_nh, int, outputs_values,
SetAllDigitalOutputs);
      if(outputs_values_set)
        outputs_values_ = outputs_values;
      else
        outputs_values_ = 5;

      {
        bool conf_outputs_values;
        int digital_output_nb;
```

```
      int configuration;
      int state;
      int mask;
      int polarity;
               if(!ParameterSetLoader(outputs_nh, "conf_outputs_values")
            .param("digital_output_nb", digital_output_nb)
            .param("configuration", configuration)
            .param("state", state)
            .param("mask", mask)
            .param("polarity", polarity)
            .all_or_none(conf_outputs_values))
           return false;

        if(conf_outputs_values){
             VCS(DigitalOutputConfiguration, digital_output_nb,
configuration, state, mask, polarity);
      }
    }

  }

  ROS_INFO("Querying Faults");
  unsigned char num_errors;
  if(!VCS_GetNbOfDeviceError(node_handle_->device_handle->ptr,
node_handle_->node_id, &num_errors, &error_code))
    return false;
  for(int i = 1; i<= num_errors; ++i) {
    unsigned int error_number;
    if(!VCS_GetDeviceErrorCode(node_handle_->device_handle->ptr,
node_handle_->node_id, i, &error_number, &error_code))
      return false;
    ROS_WARN_STREAM("EPOS Device Error: 0x" << std::hex << error_number);
  }

  bool clear_faults;
  config_nh_.param<bool>("clear_faults", clear_faults, false);
  if(num_errors > 0) {
    if(clear_faults) {
      ROS_INFO("Clearing faults");
      if(!VCS_ClearFault(node_handle_->device_handle->ptr, node_handle_-
>node_id, &error_code)) {
           ROS_ERROR("Could not clear faults");
           return false;
      }
      else
           ROS_INFO("Cleared faults");
    }
    else {
      ROS_ERROR("Not clearing faults, but faults exist");
      return false;
    }
  }

  if(!VCS_GetNbOfDeviceError(node_handle_->device_handle->ptr,
node_handle_->node_id, &num_errors, &error_code))
    return false;
  if(num_errors > 0) {
```

```
    ROS_ERROR("Not all faults were cleared");
    return false;
  }

  config_nh_.param<bool>("halt_velocity", halt_velocity_, false);

  if(!config_nh_.getParam("torque_constant", torque_constant_)) {
    ROS_WARN("No torque constant specified, you can supply one using the
'torque_constant' parameter");
    torque_constant_ = 1.0;
  }

  ROS_INFO_STREAM("Enabling Motor");
  if(!VCS_SetEnableState(node_handle_->device_handle->ptr, node_handle_-
>node_id, &error_code))
    return false;

  has_init_ = true;
  return true;
}


void Epos::read() {/////   READ
  if(!has_init_)
    return;

  unsigned int error_code;

  // Read statusword
  unsigned int bytes_read;
  VCS_GetObject(node_handle_->device_handle->ptr, node_handle_->node_id,
0x6041, 0x00, &statusword_, 2, &bytes_read, &error_code);

  int position_raw;
  int velocity_raw;
  short current_raw;
  VCS_GetPositionIs(node_handle_->device_handle->ptr, node_handle_-
>node_id, &position_raw, &error_code);
  VCS_GetVelocityIs(node_handle_->device_handle->ptr, node_handle_-
>node_id, &velocity_raw, &error_code);
  VCS_GetCurrentIs(node_handle_->device_handle->ptr, node_handle_-
>node_id, &current_raw, &error_code);
  position_ = position_raw;
  velocity_ = velocity_raw;
  ROS_INFO_STREAM(velocity_raw);
  current_ = current_raw  / 1000.0; // mA -> A
  effort_ = current_ * torque_constant_;

}

void Epos::write() {
  if(!has_init_)
    return;

  unsigned int error_code;
  if(operation_mode_ == PROFILE_VELOCITY_MODE) {
    if(isnan(velocity_cmd_))
```

```
      return;
    int cmd = (int)velocity_cmd_;
    if(max_profile_velocity_ >= 0) {
      if(cmd < -max_profile_velocity_)
           cmd = -max_profile_velocity_;
      if(cmd > max_profile_velocity_)
           cmd = max_profile_velocity_;
    }

    if(cmd == 0 && halt_velocity_) {
      VCS_HaltVelocityMovement(node_handle_->device_handle->ptr,
node_handle_->node_id, &error_code);
    }
    else {
      VCS_MoveWithVelocity(node_handle_->device_handle->ptr,
node_handle_->node_id, cmd, &error_code);
    }
  }
  else if(operation_mode_ == PROFILE_POSITION_MODE) {
    if(isnan(position_cmd_))
      return;
    VCS_MoveToPosition(node_handle_->device_handle->ptr, node_handle_-
>node_id, (int)position_cmd_, true, true, &error_code);
  }
}

void Epos::update_diagnostics() {
  diagnostic_updater_.update();
}
void Epos::buildMotorStatus(diagnostic_updater::DiagnosticStatusWrapper
&stat) {
  stat.add("Actuator Name", actuator_name_);
  unsigned int error_code;
  if(has_init_) {
    bool enabled = STATUSWORD(READY_TO_SWITCH_ON, statusword_) &&
STATUSWORD(SWITCHED_ON, statusword_) && STATUSWORD(ENABLE, statusword_);
    if(enabled) {
      stat.summary(diagnostic_msgs::DiagnosticStatus::OK, "Enabled");
    }
    else {
      stat.summary(diagnostic_msgs::DiagnosticStatus::OK, "Disabled");
    }

    // Quickstop is enabled when bit is unset (only read quickstop when
enabled)
    if(!STATUSWORD(QUICKSTOP, statusword_) && enabled) {
      stat.mergeSummary(diagnostic_msgs::DiagnosticStatus::WARN,
"Quickstop");
    }

    if(STATUSWORD(WARNING, statusword_)) {
      stat.mergeSummary(diagnostic_msgs::DiagnosticStatus::WARN,
"Warning");
    }

    if(STATUSWORD(FAULT, statusword_)) {
              stat.mergeSummary(diagnostic_msgs::DiagnosticStatus::ERROR,
```

```
"Fault");
    }

    stat.add<bool>("Enabled", STATUSWORD(ENABLE, statusword_));
    stat.add<bool>("Fault", STATUSWORD(FAULT, statusword_));
    stat.add<bool>("Voltage Enabled", STATUSWORD(VOLTAGE_ENABLED,
statusword_));
    stat.add<bool>("Quickstop", STATUSWORD(QUICKSTOP, statusword_));
    stat.add<bool>("Warning", STATUSWORD(WARNING, statusword_));

    unsigned char num_errors;
    if(VCS_GetNbOfDeviceError(node_handle_->device_handle->ptr,
node_handle_->node_id, &num_errors, &error_code)) {
      for(int i = 1; i<= num_errors; ++i) {
           unsigned int error_number;
           if(VCS_GetDeviceErrorCode(node_handle_->device_handle->ptr,
node_handle_->node_id, i, &error_number, &error_code)) {
             std::stringstream error_msg;
             error_msg << "EPOS Device Error: 0x" << std::hex <<
error_number;
             stat.mergeSummary(diagnostic_msgs::DiagnosticStatus::ERROR,
error_msg.str());
           }
           else {
             std::string error_str;
             if(GetErrorInfo(error_code, &error_str)) {
               std::stringstream error_msg;
               error_msg << "Could not read device error: " << error_str;

stat.mergeSummary(diagnostic_msgs::DiagnosticStatus::ERROR,
error_msg.str());
             }
             else {

stat.mergeSummary(diagnostic_msgs::DiagnosticStatus::ERROR, "Could not
read device error");
             }
           }
      }
    }
    else {
      std::string error_str;
      if(GetErrorInfo(error_code, &error_str)) {
           std::stringstream error_msg;
           error_msg << "Could not read device errors: " << error_str;
           stat.mergeSummary(diagnostic_msgs::DiagnosticStatus::ERROR,
error_msg.str());
      }
      else {
           stat.mergeSummary(diagnostic_msgs::DiagnosticStatus::ERROR,
"Could not read device errors");
      }
    }

  }
  else {
         stat.summary(diagnostic_msgs::DiagnosticStatus::ERROR, "EPOS not
```

```
initialized");
  }
}

void
Epos::buildMotorOutputStatus(diagnostic_updater::DiagnosticStatusWrapper
&stat) {
  std::string operation_mode_str;
  if(operation_mode_ == PROFILE_POSITION_MODE) {
    operation_mode_str = "Profile Position Mode";
    stat.add("Commanded Position",
boost::lexical_cast<std::string>(position_cmd_) + " rotations");
  }
  else if(operation_mode_ == PROFILE_VELOCITY_MODE) {
    operation_mode_str = "Profile Velocity Mode";
    stat.add("Commanded Velocity",
boost::lexical_cast<std::string>(velocity_cmd_) + " rpm");
    ROS_INFO_STREAM(velocity_cmd_);
  }
  else {
    operation_mode_str = "Unknown Mode";
  }
  stat.add("Operation Mode", operation_mode_str);
  stat.add("Nominal Current",
boost::lexical_cast<std::string>(nominal_current_) + " A");
  stat.add("Max Current", boost::lexical_cast<std::string>(max_current_)
+ " A");

  unsigned int error_code;
  if(has_init_) {
    stat.add("Position", boost::lexical_cast<std::string>(position_) + "
rotations");
    stat.add("Velocity", boost::lexical_cast<std::string>((velocity_/25))
+ " rpm");//due to our case is gear reduction 25 wheel speed is /25
    stat.add("Torque", boost::lexical_cast<std::string>(effort_) + "
Nm");
    stat.add("Current", boost::lexical_cast<std::string>(current_) + "
A");


    stat.add<bool>("Target Reached", STATUSWORD(TARGET_REACHED,
statusword_));
    stat.add<bool>("Current Limit Active",
STATUSWORD(CURRENT_LIMIT_ACTIVE, statusword_));


    stat.summary(diagnostic_msgs::DiagnosticStatus::OK, "EPOS operating
in " + operation_mode_str);
    if(STATUSWORD(CURRENT_LIMIT_ACTIVE, statusword_))
      stat.mergeSummary(diagnostic_msgs::DiagnosticStatus::WARN, "Current
Limit Active");
    if(nominal_current_ > 0 && std::abs(current_) > nominal_current_) {
      stat.mergeSummaryf(diagnostic_msgs::DiagnosticStatus::WARN,
"Nominal Current Exceeded (Current: %f A)", current_);
    }

  }
```

```
  else {
    stat.summary(diagnostic_msgs::DiagnosticStatus::ERROR, "EPOS not
initialized");
  }
}

}
```

Code of *tiera_motor1.yaml* file which provides the values of configuration parameters for the first motor.

```
# Time in seconds
# Current in amps
# position, velocity, and acceleration in device units

my_wheel_actuator1:
  actuator_name: 'wheel_actuator1'
  serial_number: '0x662080006194'
  operation_mode: 'profile_velocity'
  clear_faults: true

  motor:
    type: 10
    ec_motor:
      nominal_current: 9.84
      max_output_current: 10.500
      thermal_time_constant: 33.9
      number_of_pole_pairs: 1

  sensor:
    type: 1
    incremental_encoder:
      resolution: 500
      inverted_polarity: false

  safety:
    max_following_error: 100
    max_profile_velocity: 5000
    max_acceleration: 15000

  position_profile:
    velocity: 5000
    acceleration: 8000
    deceleration: 9000

  position_regulator:
    gain:
      p:  1959
      i:  7380
      d:  2450
    feed_forward:
      velocity:  25829
      acceleration:  834

  velocity_profile:
    acceleration: 2000
    deceleration: 2000
    #window:
    #      window:
    #      time:

  velocity_regulator:
    gain:
      p:  6100
```

```
      i:  1306
#    feed_forward:
#      velocity:  '0x64E5'
#      acceleration:  '0x0342'

  current_regulator:
    gain:
      p:  248
      i:  46

  outputs:
    outputs_values: 5
    conf_outputs_values:
      digital_output_nb: 4
      configuration: 12
      state: 1
      mask: 1
      polarity: 1
```

APPENDIX IX

Code of *tiera_track_robot.launch* file which initializes EPOS2 70/10 controllers.

```
<launch>
  <param name="robot_description" textfile="$(find
epos_hardware)/launch/example_
test_mcs.urdf" />
  <node name="epos_hardware" pkg="epos_hardware"
type="epos_hardware_node" args="my_wheel_actuator1 my_wheel_actuator2
my_wheel_actuator3 my_wheel_actuator4">
    <rosparam command="load" file="$(find
epos_hardware)/launch/tiera_motor1.yaml" />
    <rosparam command="load" file="$(find
epos_hardware)/launch/tiera_motor2.yaml" />
    <rosparam command="load" file="$(find
epos_hardware)/launch/tiera_motor3.yaml" />
    <rosparam command="load" file="$(find
epos_hardware)/launch/tiera_motor4.yaml" />
  </node>

  <node name="controller_spawner" pkg="controller_manager" type="spawner"
respawn="false"       output="screen"  args="--timeout 120
joint_state_controller1 velocity_controller1 joint_state_controller2
velocity_controller2 joint_state_controller3 velocity_controller3
joint_state_controller4 velocity_controller4 "/>
    <param name="velocity_controller1/type"
value="velocity_controllers/JointVelocityController" />
    <param name="velocity_controller1/joint" value="joint_wheel_BL" />
    <param name="velocity_controller2/type"
value="velocity_controllers/JointVelocityController" />
    <param name="velocity_controller2/joint" value="joint_wheel_BR" />
    <param name="velocity_controller3/type"
value="velocity_controllers/JointVelocityController" />
    <param name="velocity_controller3/joint" value="joint_wheel_FR" />
    <param name="velocity_controller4/type"
value="velocity_controllers/JointVelocityController" />
    <param name="velocity_controller4/joint" value="joint_wheel_FL" />

    <param name="joint_state_controller1/type"
value="joint_state_controller/JointStateController" />
    <param name="joint_state_controller1/publish_rate" value="50" />

    <param name="joint_state_controller2/type"
value="joint_state_controller/JointStateController" />
    <param name="joint_state_controller2/publish_rate" value="50" />
    <param name="joint_state_controller3/type"
value="joint_state_controller/JointStateController" />
    <param name="joint_state_controller3/publish_rate" value="50" />
    <param name="joint_state_controller4/type"
value="joint_state_controller/JointStateController" />
    <param name="joint_state_controller4/publish_rate" value="50" />

  <node name="robot_state_publisher" pkg="robot_state_publisher"
type="state_publisher" />

</launch>
```

APPENDIX X

Code of *tiera_track_station.launch* file which initializes Xbox controller in ROS environment.

```
<launch>
<node name="joy_node" pkg="joy" type="joy_node">
 <param name="dev" type="string" value="/dev/input/js3" />
 <param name="deadzone" value="0.1" />
 <param name="coalesce_interval" value="0.05" />
</node>

<node    name="lut_controller"    pkg="lut_track"    type="lut_controller"
output="screen" launch-prefix="gnome-terminal --command"/>

  <node name="kinematics_server" pkg="lut_track" type="kinematics_server"
output="screen" launch-prefix="gnome-terminal --command"/>

</launch>
```