|  |  |  |
| --- | --- | --- |
| **Автономная некоммерческая организация высшего образования**  **«Университет Иннополис»** |  | **Autonomous noncommercial organization of higher education**  **«Innopolis University»** |

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**ПО НАПРАВЛЕНИЮ ПОДГОТОВКИ**

**09.03.01 – «ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА»**

**GRADUATE THESIS**

**MAJOR: «COMPUTER SCIENCE»**

|  |  |  |
| --- | --- | --- |
| **Тема** |  | **Разработка инструментов управления роботом DARwIn OP2 (Dynamic Antropomorphic Robot with Intelligence — Open Platform — Динамичный антропоморфный робот с интеллектуальной платформой)** |
|  | | |
| **Topic** |  | **DARwIn OP2 control tools development (Dynamic Antropomorphic Robot with Intelligence – Open Platform)** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Работу выполнил /  Thesis is executed by |  | **Сотниченко Роман Евгеньевич**  **Sotnichenko Roman Yevgenyevich** |  | подпись / signature |
|  |  |  |  |  |
| Научный руководитель /  Thesis supervisor |  | **Климчик Александр Сергеевич**  **Klimchik Alexander Sergeevich** |  | подпись / signature |
|  |  |  |  |  |

Иннополис, Innopolis, 2017

**Table of contents**

**1 Introduction**

The situation with the development of robotics is such that it is not possible to stop progress in this area. Robots have already became the part of a new industrial revolution, the main features of which are robotization of manufacturing and wide embedding of additive technologies such as 3D-printing. These processes complement each other, because emedding of 3D-printing significantly decreases the volumes of assembly works and amount of types of machining. Every year more and more factories are getting automized, and now a factory, which employs 20-30 people, and all other job is being done by robots — is not unusual.

Robotics becomes a promoter of any industry, because it contributes to research and development, production of precision mechanics, electrical engineering, electronics, optics, composite materials, etc.

Deveolpment of robotics allows to solve different social problems, like limiting the migration of low-skilled labor, caring for the elderly, reducing human losses in military conflicts and transport.

The progress of robotics is quite obvious, that's why in the next ten years one can expect the following events:

- Appearance and the beginning of mass introduction of the robotized transport, a transport without the human-driver. This process does not go as fast as one would like, but in a decade it will reach the point when it will finally be accepted by society in developed countries.

- In the area of military robotics, unmanned aircraft (UAV) will continue to displace pilots from the Air Force. Most likely, the ratio of aircraft will tend to a ratio of 80:20 in favor of drone. Similarly, the replacement of servicemen by robots in all other types of armed forces wil also increase.

- A stable market of service robots, primarily households, will be formed, on which such functions as cleaning and security of housing, childcare, cooking and leisure of family members will form. One should expect the appearance of all sorts of robotic nurses, training robots. In this direction, there is a huge amount of development, and, we think, in the next five or ten years almost every family will acquire at least one service robot of one type or another.

Social robotics — a field that designs robots with behavior systems inspried by how humans communicate with each other. Social roboticists might incorporate posture, timing of motion, prosody of speech, or reaction to people and environments into a robot's behavioral reportoire to help communicate the robot's state or intentions. The benefit of such systems is that they enable bystanders and interaction partners to undrestand and interact with robots without prior training. This opens up new applications for embodied machines in our everyday lives.

The implications of this social dimension to robotics have reinforced the design and building of human-like robots (Honda Asimo, PINO) [1]. In the extreme, the building of strongly human-like robots constrains the functionality that a robot could otherwise have. The role of anthropomorphism in robotics is not to build an artificial human, but rather to take advantage of it as a mechanism through which social interaction can be facilitated. Once robots come out of the washing machine and start moving around our physical and social spaces, their role and our dealings with them will change significantly. It is in emracing their inherent advantage in being machines rather than seeing this as a disadvantage that will lead to their success.

Antrhropomorphism is prevalent in robotics because of one's tendency to need such familiarity. Robotics research focuses in building humanoids not because a humanoid is the most efficient design for any given task, but rather because of one's tendency and need to anthropomorphise. Searle already points out the dangers present through confounding reasoning and rationalising because of this tendency to anthropomorphise (1992). Effectively, anthropomorphism obstructs the fact that the human form is not the ideal for a machine. However, objectively, the humanoid form best helps is interact with machines because of the fact that we anthroporphise. But this does not necessarily mean that one should build seamlessly perfect-looking human-machines as anthropomorphism alows roboticists be heuristic in robot design (i.e. cartoon characters).

The application scenarios of application of anthropomorphic robots can include a museum guide, a corporate building butler, or a post office clerk. But they can do not only such usual routine job. When people are in danger, robots also come in. Some can handle hazardous materials. If some valve needs to be shut off manualy in a room full of poison gas in a burning building, robot can help.

Children with disorders on the autism spectrum often have a tough time learning socual skills. Some are actually frightened or disturbed when dealing with other people, or just have problems learning how to pay attention to those around them. Korean robot manufacturer «ROBOTIS» has designed a robot [2] called DARwIn (Dynamic Anthropomorphic Robot with Intelligence) that is being tested in Chung Hyuk Park's (an assistant professor of biomedical engineering at George Washington University) project to help children with autism to be more engaged with society [3].

Beginning in 2012 ROBOTIS and IEEE ICRA has sponsored the DARWIN-OP Humanoid Application Challenge, held at the ICRA conference. The competition encourages participants to solve novel problems using DARWIN-OP and present their findings at the conference. The projects were: ice skating/ice hockey (2012), case-based reasoning, learning from demonstration(2013), Alpine and cross-country skiing (2015). Also DARWIN-OP robot is currently used by several teams in the RoboCup kids-size competition (humanoid robots 40-60 cm tall) and have shown excellent performance by the teams using this platform.

In the area of fighting sport there are competitions for robots. «Robot combat» is a hobby in which two or more custom-built machines use varied methods of destroying or disabling the other, «BattleBots» is an American robot competition television series. Robo-One — autonomous biped fighting tournament.

The aim of this graduate thesis — «teach» robot DARwIn to fight with another DARwIn and create a remote controller for it.

The things that DARwIn should be able to do to are walking (backward and forward, backward and forward with turn, turning around), and combat actions (arm pushes an punches, leg kicks, blocks). Also it should be able to get up from backward and forward fails. For gait pattern generation coupled oscillator model is used [4].

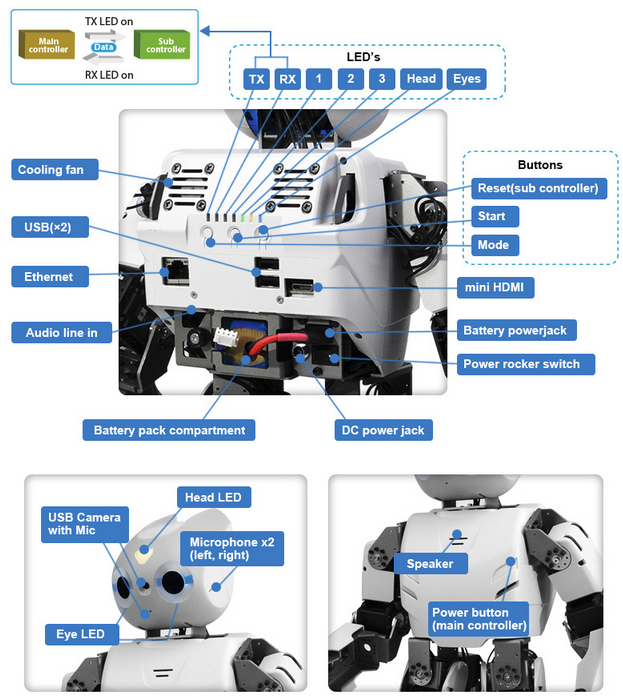
Remote controller for the robot should be convenient to send different commands and constantly maintain a connection with the least number of disconnections and delays. Tablet with Android OS (Samsung Galaxy Tab S2) is chosen for that, because Android SDK provides suitable tools and libraries to make a remote controller for robot.

**2 System specifications**

ROBOTIS OP2 (DARwIn-OP2) is an affordable, miniature-humanoid-robot open platform with advanced computational power, sophisticated sensors, high payload capacity, and dynamic motion ability to enable many exciting research, education and outreach activities.

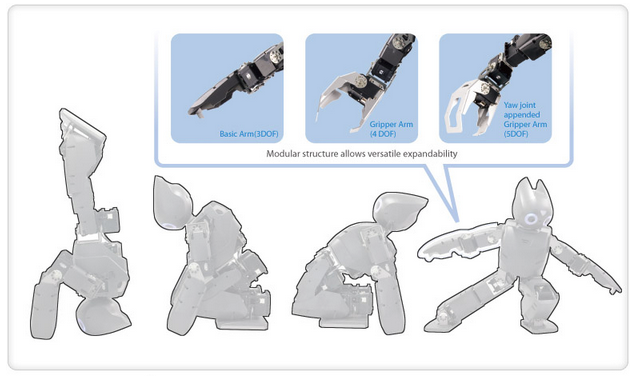
«ROBOTIS OP2» is a new version of «ROBOTIS OP» that was formerly known as «DARwIn-OP».

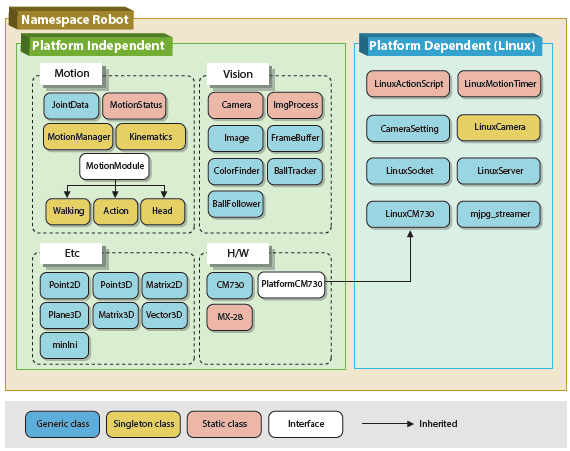
High performance and advanced features:

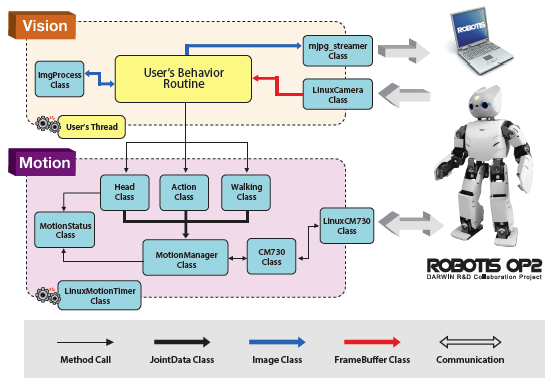
**Figure 1.** Standard PC-based robot with convenient interfaces

* default walking speed: 24.0 cm/sec (9.44 in/sec) 0.25 sec/step - user modifiable gait;
* default standing up time from ground: 2.8 sec (from facing down) and 3.9 sec (from facing up) - user modifiable speed;
* built-in PC: Intel Atom N2600 @1.6 GHz dual core, RAM 4GB DDR3, 32GB mSATA;
* management controller (CM-740): ARM CortexM3 STM32F103RE 72MHz;
* 20 actuator modules (6 DOF leg x2+ 3 DOF arm x2 + 2 DOF neck);
* actuators with durable metallic gears (DYNAMIXEL MX-28T);
* self-maintenance kit (easy to follow steps and instructions);
* standby mode for low power consumption;
* 1Mbps high-speed Dynamixel bus for joint control;
* 1800mAh LIPO Battery (30 minutes of operations), charger, and external power adapter (Battery can be removed from robot without shutting down by plugging in external power before removal);
* versatile functionality (can accept legacy, current, and future peripherals);
* 3-axis gyro, 3-axis accelerometer, button x3, detection microphone x2.

**Figure 2.** Simple and clean architecture

**Figure 3.** Efficient and versatile modular configuration

**Figure 4.** ROBOTIS OP2 framework class library diagram

**Figure 5.** ROBOTIS OP2 data flow diagram

**3 Background theory**

**3.1 DARwIn-OP2 Kinematics**

**3.1.1 DARwIn-OP2 Parameters**

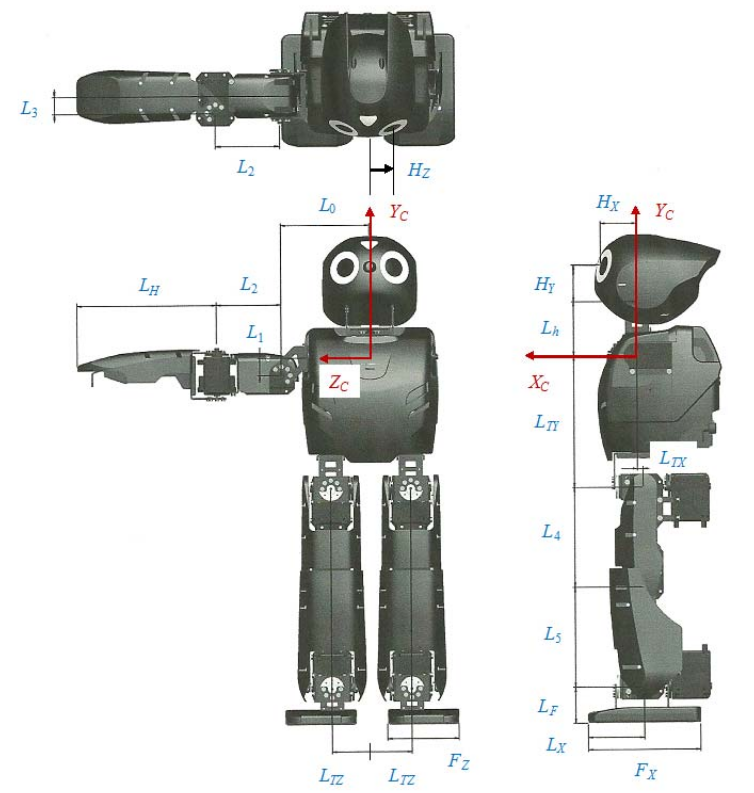
The joint angle values will be denoted by according to the numbering scheme in Figure 6. The pose shown is for all zero angles.

**Figure 6.** DARwIn-OP2 servomotor numbering

For notational simplicity in Cartesian coordinate frame definition, frame numbers etc. will be etc. will be recycled for the five serial chains. Thus, there will be five axes, four axes, two axes, and so on. This is to avoid the onerous notion of etc. (responding right arm and left leg).

According to Craig (2005) notation for the Denavit-Hartenberg Parameters (1995), joint angleis supposed to rotate about R-joint axisAgain for simplicity in notation, this rule will be violated, in order to number the joint angles as shown in Figure 6.

Below (tables 1 - 5) there are various link lengths for the DARwIn-OP2 humanoid robot that are needed to solve forward and inverse kinematics equations. Refer to the lengths definitions in Figure 7 for these parameters.

**Figure 7.** DARwIn-OP2 robot length definitions

|  |  |
| --- | --- |
| **Length** | **Value (mm)** |
|  | 50.5 |
|  | 82.0 |
|  | 5.0 |
|  | 122.2 |
|  | 37.0 |

**Table 1.** Torso lengths

|  |  |
| --- | --- |
| **Length** | **Value (mm)** |
|  | 33.2 |
|  | 34.4 |
|  | 22.5 |

**Table 2.** Head lengths

|  |  |
| --- | --- |
| **Length** | **Value (mm)** |
|  | 16.0 |
|  | 60.0 |
|  | 16.0 |
|  | 129.0 |

**Table 3.** Right and left arm lengths

|  |  |
| --- | --- |
| **Length** | **Value (mm)** |
|  | 93.0 |
|  | 93.0 |
|  | 33.5 |

**Table 4.** Right and left leg lengths

|  |  |
| --- | --- |
| **Length** | **Value (mm)** |
|  | 104.0 |
|  | 15.0 (10.0 front) |
|  | 66.0 |
|  | 52.0 |
|  | 23.0 |

**Table 5.** Right and left foot lengths

In table below there are joint angle limits for DARwIn-OP2 humanoid robot. These are specific hardware design values associated with the DH parameters.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Joint** | **Joint name** | **Axis** |  |  |
| **Right arm** | | | | |
| 1 | shoulder pitch |  | -250 | 250 |
| 3 | shoulder roll |  | -100 | 100 |
| 5 | elbow |  | 0 | 160 |
| **Left arm** | | | | |
| 2 | shoulder pitch |  | -250 | 250 |
| 4 | shoulder roll |  | -100 | 100 |
| 6 | elbow |  | -160 | 0 |
| **Right leg** | | | | |
| 7 | hip yaw |  | -150 | 45 |
| 11 | hip roll |  | 0 | 60 |
| 9 | hip pitch |  | -100 | 30 |
| 13 | knee |  | 0 | 130 |
| 17 | ankle pitch |  | -60 | 60 |
| 15 | ankle roll |  | -30 | 60 |
| **Left leg** | | | | |
| 8 | hip yaw |  | -45 | 150 |
| 12 | hip roll |  | -60 | 0 |
| 10 | hip pitch |  | -30 | 100 |
| 14 | knee |  | -130 | 0 |
| 18 | ankle pitch |  | -60 | 60 |
| 16 | ankle roll |  | -30 | 60 |
| **Head** | | | | |
| 19 | head pan |  | -150 | 150 |
| 20 | head tilt |  | -60 | 30 |

**Table 6.** DARwIn-OP2 joint limits

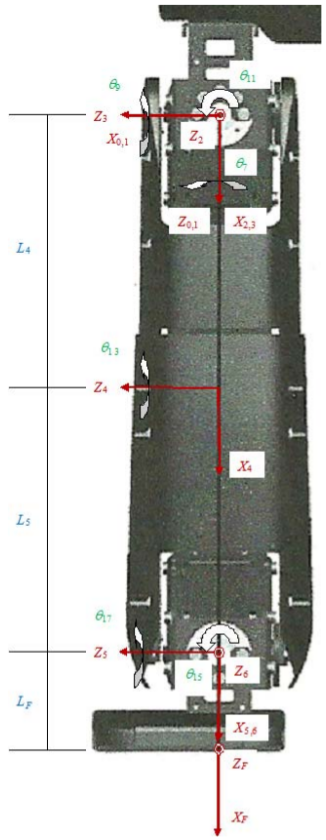
**3.1.2 DARwIn-OP2 Forward Pose Kinematics**

In general, the forward pose kinematics (FPK) problem for a serial-chain robot is stated: Given the joint values, calculate the pose (position and orientation) of the end frame of interest. For serial-chain robots, the FPK problem set up and solution is straight-forward. It is based on substituting each line of the Denavit-Hartenberg parameters table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 1 | 0 | 0 | 0 |  |
| 2 | 90 | 0 | 0 |  |
| 3 | 90 | 0 | 0 |  |
| 4 | 0 |  | 0 |  |
| 5 | 0 |  | 0 |  |
| 6 | -90 | 0 | 0 |  |

**Table 7.** Six-DOF right leg DH parameters

into the equation below (Craig, 2005), giving the pose of framewith respect to its nearest neighbor frameback along the serial chain:

**Figure 8.** Six-DOF right leg coordinate frame

The equation above represents pose (position and orientation) of framewith respect to frame by using a 4x4 homogeneous transformation matrix. The upper left 3x3 matrix is the rotation matrix giving the orientation and the upper right 3x1 vector is the position vector.

Then transformation equations are used to find the pose of the overall end-frame of interest with respect to the base reference frame, to complete the FPK solution for each serial chain.

**3.1.3 Six-DOF Right Leg FPK Expressions**

The statement of the FPK problem for the six-DOF right leg serial chain of the DARwIn-OP humanoid robot is:

Given: calculate and

Where is the right-leg end-effector (foot) frame and is the DARwIn chest reference frame. Substitute each row of the DH parameters in Table 7 into the equation for to obtain the six neighboring homogeneous transformation matrices as a function of the joint angles.

Where the following abbreveations were used: for

Now substitute these three neighboring homogeneous transformation matrices into the following homogeneous transform equation to derive the FPK result.

Since there are three parallelaxes (3, 4, 5) in the right leg, we can group the above matrix multiplications as follows for a significant simplification.

Where:

where the abbreviations were used.

The right FPK result is:

where:

The basic right-leg FPK result is To calculate the pose of the right-leg end-effector frame with respect to the DARwIn chest reference frame the following transform equation is used:

The overall transform equation above can be evaluated numerically. Constants and were given earlier.

Note that andare not evaluated by any row in the DH parameter table, since there is no variable associated with these fixed homogeneous transformation matrices based on constant lengths and orientation. Instead, they are determined by inspection, using the rotation matrix and position vector components of the basic homogeneous transformation matrix definition.

**3.1.4 DARwIn-OP2 Inverse Pose Kinematics**

In general, the inverse pose kinematics (IPK) problem for a serial-chain robot is stated: Given the pose (position and orientation) of the end frame of interest, calculate the joint values. For serial-chain robots, the IPK solution starts with the FPK equations. The solution of coupled nonlinear algebraic equations is required and multiple solution sets generally result. The FPK examples can also serve as IPK examples when the input is reversed.

**3.1.5 Six-DOF Right Leg IPK Solution (partial)**

The statement of the IPK problem for the six-DOF right leg serial chain of the DARwIn-OP humanoid robot is:

Givencalculate

Whereis the right-leg end-effector foot frame andis the DARwIn chest reference frame.

We can first simplify the givento using the following transform equation:

Where constant transform matricesand were given in the FPK section for the right leg. Let the following symbols represent the numerical values for derived from the originally-given

These numerical values are equated to the FPK solution as a function of the six unknown joint angles:

The above represents 16 equations (4 trivial) in the six unknown joint angles. The three position vector component equations are all independent, but there are only three independent equations among the 9 equations of the rotation matrix. The equations are coupled and nonlinear (trenscendential).

Looking at the FPK terms given earlier, we need some simplification prior to solving these equations. There are three possibilities:

1. Notice that the hip is spherical (i.e. framesshare a common origin point);

2. FPK already took advantage of the three parallel axes (last hip kneeand first ankle);

3. Since the hip is spherical and the 2-DOF ankle also rotates about a common point, the distance from the hip center to the ankle conter is only a function of one joint angle,The third simplification will be expoiled first.

Note this is equivalent to the Law of Cosines, looking at the plane containing the hip, knee, and both leg lengthsand

**3.2 Robot Operating System (ROS)**

The ROS is a framework used for writing software for robots. It is a set of conventions, tools, and libraries designed to simplify the creation of complex and robust robot behavior across a wide variety of robotic platforms.

Creating robust, general-purpose robot software is hard. Problems that seem trivial to humans often vary wildly between different tasks and environments, from the robot's perspective.

ROS was built from the ground up to encourage collaborative robotics software development. For example, one individual is an expert in different walking algorithms for biped robots and could contribute such an algorithm for using it in humanoid robots. Some laboratory might have experts at computer vision approaches that works well for building 3D maps. And another group might have experts at using maps in navigation. Those three groups of experts can collaborate and build a good robot that can bypass different obstacles.

ROS provides standard operating system services such as hardware abstraction, low-level device control, frequently used functions implementation, messaging among processes and package management. The library is focused on Unix-like systems (Ubuntu is included in the list of «supported», while other options, such as Fedora and Mac OS X, are considered «experimental»).

ROS has two main «sides»: the sides of the ros operating system, as described above and ros-pkg, a set of user-supported packages (organized into sets called the stack) that implement various functions of robotics: SLAM, planning, perception, modelling, etc.

ROS is based on the graphs architecture, where data processing occurs in nodes that can receive and transmit messages to each others by the means of topics.

ROS can be compared to a UNIX system. Do not write one big programm, but implement a few small ones that can together solve the task. This is how nodes work in ROS. But if in UNIX programs interact with each other through pipes, in ROS, the programs (nodes) interact with each other through the topics. In UNIX programs send data to each other. In ROS — messages. Thus, one can take different programs and get a quick result — just by combining them together. The same is in ROS, but instead of the usual computer utilities, there are programs for robots. For example, instead of the file search program — node for finding the location. Instead of the program for finding the right line — node for searching for an object on the camera image. Instead of the file transfer command — node for manipulator control.

**4 Implementation**

The task of creating the fighting robot is separated into smaller subtasks. Walking, arm strikes, and remote control.

**4.1 Remote control**

**4.1.1 Connection**

To send commands and get responses from DARwIn-OP2 one of the standards of wireless communication was chosen. Wired connection is not appropriate, since ethernet cable, sticking out of robot, affects its balancing for the worse and may damage robot's ethernet port.

The choise came between two standards — Bluetooth and WiFi. The factors for chosing WiFi connection were:

1. less latency (WiFi ~ 150ms, Bluetooth ~ 200ms);
2. higher bit-rate (WiFi ~ 600 Mbps, Bluetooth 2.1 Mbps);
3. some physical restrictions. DARwIn-OP2 doesn't have a Bluetooth, so the external one is needed. But since robot may fall on its back, we can't use USB Bluetooth adapter, because that USB port or adapter itself that sticks out may be damaged, like the ethernet port.

**4.1.2 Remote controller**

For creating the remote controller Android Studio (free integrated development environment for working with Android platform) with Java and Android software development kits — are free tools for building apps on every type of android device.

Java Development Kit (JDK) includes the Java compiler, standard Java class libraries, examples, documentation, various utilities and the Java executive system JRE. JDK does not include an integrated development environment for Java.

Android SDK — is a universal tool for developing mobile applications for the Android operating system. A distinctive feature of ordinary editors for writing codes is the availability of broad functionality, which allows to run testing and debugging of source codes, evaluate the application performance in compatibility mode with different versions of Android OS and watch the result in real time. Supports a large number of mobile devices, among which are: mobile phones, tablet computers, smart glasses, modern cars with on-board computers on Android OS, advanced TVs, special wrist watches and many other mobile gadgets.

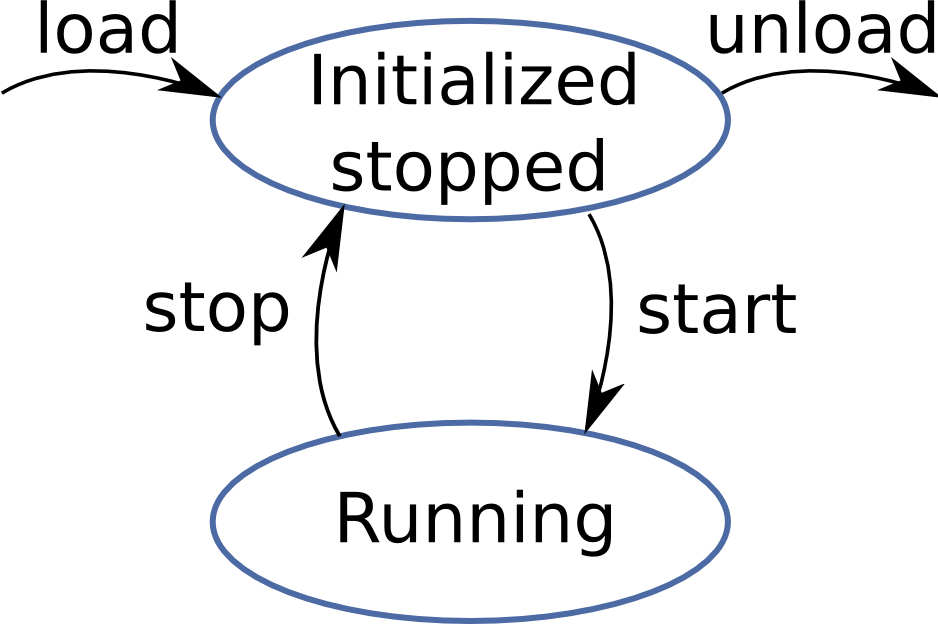
Extra Java library was used to create ssh connection between Android device and DARwIn-OP2. «Java Secure channel» (JSch) — is a pure Java implementation of SSH2. JSch allows to connect to an sshd server and use port forwarding, X11 forwarding, file transfer, etc., and one can integrate its functionality inte his own Java programs. In this graduate thesis JSch was used to open shell channel between Android device and robot.

For the remote controller Samsung Galaxy Tab S2 SM-T813 was used.

The main window of application for robot control consist of the next parts:

1. terminal (Figure 9). This part displays response of commands that are sent to robot via SSH channel. One can execute a command by placing it into the input field and clicking «EXECUTE CMD» button. «CLEAR LOG» button clears the output window and «CLEAR CMD» clears the cmd input field. To enable the input field one must click «CONNECT CMD» button and wait until channel is opened. After that a command can be executed.
2. panel with connection settings (ip of robot, port, user, and password — for SSH connection, robot must be connected to the same network as Android device). Also there are buttons:

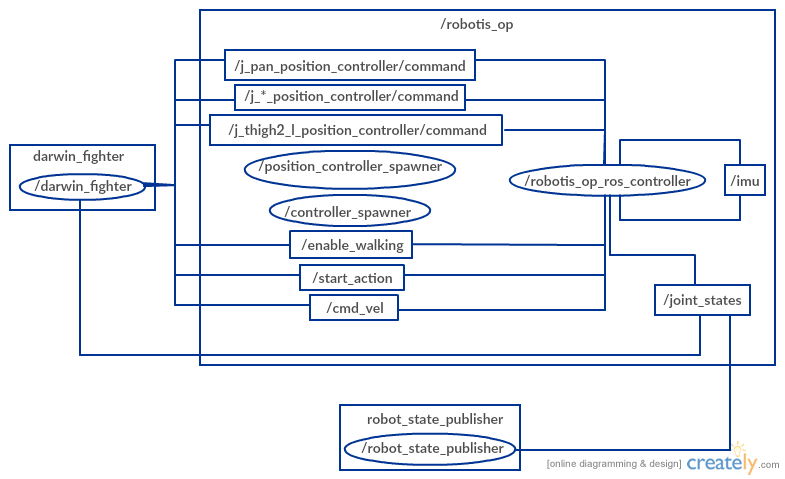
* «AUX DEMO» — searches the demo process that starts automatically when the robot is switched on;
* «KILL DEMO» — kill demo process. It should be killed so that whole\_robot\_launch node could be launched;
* «ROSCORE» — button that launches roscore process — a collection of nodes and programs that are pre-requisites of a ROS-based system. It must be running in order for ROS nodes to communicate. Roscore will start up the ROS Master, that provides naming and registration services to the rest of the nodes in the ROS system. It tracks publishers and subscribers to topics. The role of the Master is to enable individual ROS nodes to locate one another. Once these nodes have located each other they communicate with each other peer-to-peer. The Master also provides the Parameter Server — a shared, multi-variate dictionary that is accessible via network APIs. Nodes use this server to store and retrieve parameters at runtime. As it is not designed for high-performance, it is best used for static, non-binary data such as configuration parameters. It is meant to be globally viewable so that tools can easily inspect the configuration state of the system and modify if necessary. Also roscore starts up «rosout» — console log reporting mechanism in ROS;
* «LAUNCH» — executes script which runs nodes responsible for camera («robotis\_op\_camera» package) and control («robotis\_op\_ros\_control» package). Also it starts up «joint\_state\_publisher» node, which publishes «sensor\_msgs/JointState» messages for a robot. The package reads «robot\_description» parameter, finds all of the non-fixed joints and publishes a «JointState» message with all those joints defined. The message includes the next fields: «string[] name», «float64[] position», «float64[] velocity» and «float64[] effort». «robotis\_op\_ros\_control» launch file includes another launch file «load\_position\_controllers» that initializes node «controller\_manager». This node provides a realtime-compatible loop to control a robot mechanism, which is represented by a «hardware\_interface::RobotHW» instance. The «controller\_manager» provides the infrastructure to load, unload, start and stop controllers. All of 20 controllers, each is responsible for it's joint and has a unique name «j\_shoulder\_r\_position\_controller», «j\_pelvis\_r\_position\_controller», etc. This spawner node («controller\_manager») will get controllers loaded and started, and will keep running while the controllers are up. When spawner is killed, it will automatically stop and unload all controllers it initially started (Figure 9);

**Figure 9.** Controller manager

* «PUPPET» — button launches «darwin\_fighter.py» python script. It has one main class «Darwin» which has joint command subscribers («joint\_state») and publishers («\*\_position\_controller/command», «cmd\_vel», «start\_action»). From the script a node is started up, which is running a loop listening for different strings or characters input. Depending on the input, a message is sent to one of the above topic. For example, character «b» increases angular and linear speeds of DARwIn-OP2 by 10% by sending a message «geometry\_msgs/Twist» (geometry\_msgs/Vector3 linear {float64 x, float64 y and float64 z}, geometry\_msgs/Vector3 angular {float64 x, float64 y and float64 z}) to «cmd\_vel» topic, character «1» sends message «13» to «start\_action» topic which means that robot will do kick with left leg, character «a» makes DARwIn do uppercut strike by publishing to arm's joints («\*\_position\_controller/command») an angle («std\_msgs/Float64»), and so on;
* «^C» buttons — clicking to which near one of the buttons for openning SSH connection «ROSCORE», «LAUNCH», «PUPPET» or «CONNECT CMD» closes that connection;

**Figure 10.** The main window of controller Android application. 1 - terminal, 2 - connection and launch panel, 3 - control panel

1. control panel — contains control buttons. Click on every button sends a character or a string via SSH channel to «darwin\_fighter.py», that processes the input and makes robot act.

**Figure 11.** Nodes and topics after running «darwin\_fighter.py» script

**4.2 «darwin\_fighter» node**

After «darwin\_fighter.py» script is launched by clicking «PUPPET» button, it initializes a new node with name «darwin\_fighter». This node sends messages to topics «enable\_walking», «start\_action», «cmd\_vel» and other 20 nodes «j\_\*\_position\_controller/command» each responsible for one of the DARwIn's joints.

**4.2.1 Walking**

aoeu

**4.2.2 Messages generation for arm moves**

To do some action with hand the next steps are performed. Knowing the joint limits described in Table 6, we can send messages with the value of an angle to a topic, responsible for that joint. For example, we've got input string that makes DARwIn perform an uppercut with right arm. Knowing the limits of angles of each joint of 3-DOF arm, we choose the final position of this arm at the end of the hit as we want. We initialize dictionary of 6 entries (3 for the right hand and 3 for the left): 'j\_high\_arm\_r': 'j\_low\_arm\_r': 'j\_shoulder\_r': 'j\_high\_arm\_l':'j\_low\_arm\_l':'j\_shoulder\_l':It's more convenient to have zero angles of joints as shown in figure 6. That's why we have a function for transforminig angles. It transforms angles in the next way:

If time that is given to make the whole move until the end position is 2 seconds, ratio coefficient is evaluated:

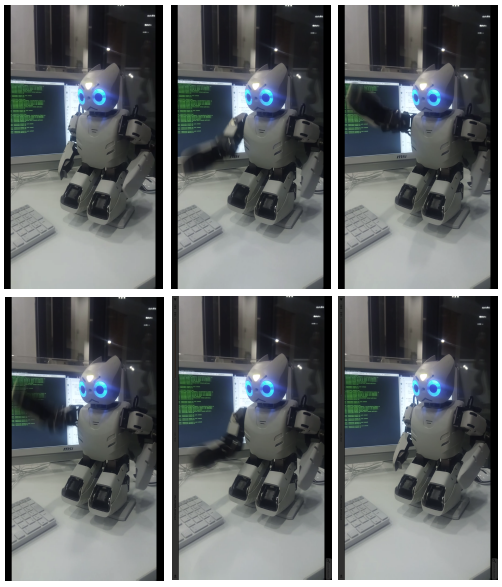
and input angles are interpolated into a new ones:

in every time moment and are published as a message to the topic with the same name.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| start\_angle (t = 0.0s) | angle value (t = 0.2s) | angle value (t = 0.4s) | angle value (t = 0.6s) | angle value (t = 0.8s) | stop\_angle (t = 1.0s) |
|  |  |  |  |  |  |

**Table 8.** Right shoulder angle tabulation in right uppercut with delay 1s and ratio 200ms

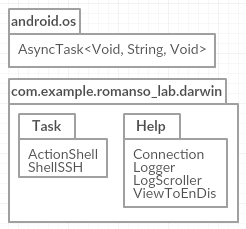
oaeu

**Figure 12.** DARwIn-OP2 right uppercut

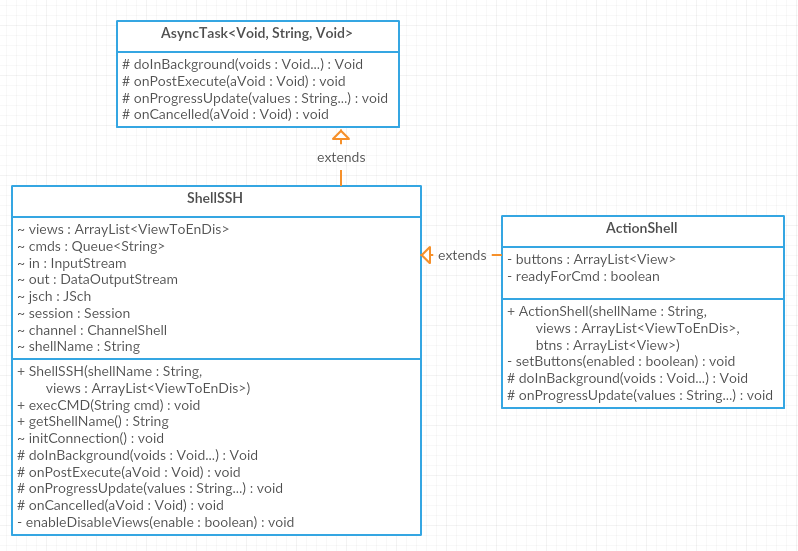
**4.3 Android application architecture**

Android application consists of several classes, each has its own purposes:

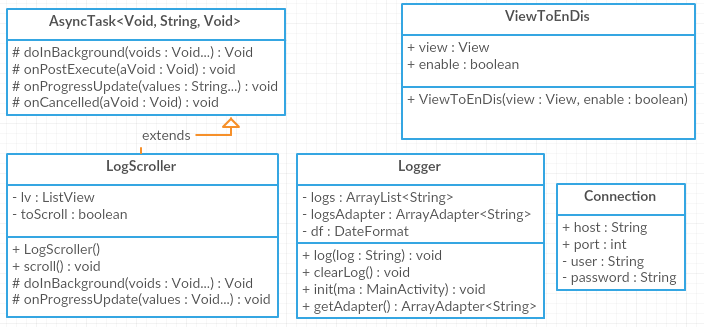
1. «ShellSSH» — object of this class creates session with robot and opens SSH channel for sending commands. Instance of «ShellSSH» creates «JSch» object, which starts a new session with robot using data from the input fields (username, host, port, password). After session has been connected, shell channel opens. There is another option — to use exec channel. But shell was chosen. The difference between these two is that exec channel creates a new shell instance and closes it after command has been executed. We use shell schannel because we need to detect completion of a command («roscore», «roslaunch robotis\_op\_onboard\_launch robotis\_op\_whole\_robot.launch», «killall demo»). After the shell channel has been opened, asynctask (Android class for working with multithreading) starts a new thread that begins a loop, which ends only if channel is closed. In this loop there is a condition for checking emptyness of commands queue. If the queue is not empty it pops the first command and executes it. And in asynktask's background there is a loop that also stops when the channel is closed. This loop reads responses coming out of the shell channel. It writes output bytes, if there are any, into a temporary byte array and then passes this array translated into a string to Logger's log method;

**Figure 13.** Classes and packages UML diagram

1. «ActionShell» — this is an extension class of «ShellSSH». The only difference between this class and the extended one is that it opens a shell channel after «PUPPET» button has been clicked. «darwin\_fighter.py» is an interactive script which waits for user's input in a loop. And depending on the input, node, that is initialized in this script, sends one of the messages to robot controller node. Also it has an array list of control panel's buttons (only of those which make DARwIn do some action — walk, hit, stand up, etc), because here it disables all of them in order to user couldn't click on any of them and add a command for action into a queue;

**Figure 14.** Shell classes hierarchy UML diagram

1. «MainActivity» — this class works and processes actions done on all of the view elements (buttons, edit texts and list view). Main activity class initializes instances of other classes («Logger», «LogScroller», «ShellSSH», «ActionShell»). This class sends data to each connection class about what buttons must be enabled or disabled after clicking on each of them;
2. four help classes — «Logger» class is used for displaying logs in terminal. It has method «log(String log)» that adds input string into logs array list and notifies array adapter that the list has changed in order to update terminal. Method «clearLog()» removes all data from logs array list and notifies array adapter that will clear terminal. «LogScroller» class is used to scroll terminal logs. It runs another thread that continiously checks for array adapter update, and if it is, «setSelection(int position)» method of «ListView» object is called, that scrolls this list view. «ViewToEnDis» class is used to enable or disable buttons that are present in main window of the application. This is an abstract data structure, that has link to a button and a flag that denote button's state (to enable or to disable in one of the «Shell» asynctasks). «Connection» class has four fields used for connection that are public «String host», «int port», «String user», «String password».

**Figure 15.** Help classes UML diagram

**Conclusion**

**List of used literature**

1. Antropomorphism and robotics (Brian R. Duffy) (<http://www.prism.ucd.ie/publications/pub2002/AISB02-Duffy.pdf>)

2. https://en.wikipedia.org/wiki/DARwIn-OP

3. darwin autism - <https://www.usatoday.com/story/news/politics/2016/04/26/robot-helps-social-skills-autistic-children/83554568/>

4. Gait Pattern Generation and Stabilization for Humanoid Robot Based on Coupled Oscillators

<http://www.forbes.ru/mneniya-opinion/idei/273731-k-chemu-privedet-razvitie-robototekhniki-v-blizhaishie-10-let>

<https://www.brookings.edu/research/how-humans-respond-to-robots-building-public-policy-through-good-design/>

diff btw bluet and wifi:

<https://www.engineersgarage.com/contribution/difference-between-bluetooth-and-wifi>

**Application**