BMCC

Borough of Manhattan Community College™

HONORS PROJECT

Controlling a humanoid robot using non-verbal communication

Student: Mehmet Sahin

Mentor: Dr. Mohammad Azhar

Course: CSC 211

Semester: Spring 2018

Date: 5/01/18

ABSTRACT

The popular humanoid robot, NAO, developed by Aldebaran Robotics, has been used in many applications such as promoting and teaching computational thinking, engineering, coding, and has been used in robotics competitions. The research in human-robot communication is in its infancy. In this paper, we will explore how a humanoid NAO robot can solve a maze game – an experimental environment that can be applied to real-world challenges – with the help of a human through visual signs such as NAOMarks and non-verbal communication (e.g., body motion) capturing using a Microsoft Kinect.

Key words: NAO Humanoid Robot, Microsoft Kinect, Human-Robot interaction, Human-Robot Communication.

1. INTRODUCTION

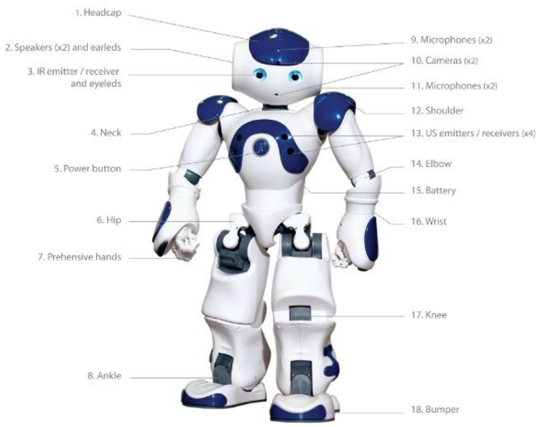
The field of robotics is generating increased attention in recent years, including the subfield of control theory and its application in the control of humanoid robots through signals [1]. In terms of human-robot interaction, many robots are able to recognize pictures and voice commands, but few can sufficiently identify non-verbal communication such as human body gestures [2]. In this project, we created a game prototype to enable human-robot interaction using visual objects. We use the NAO robot as the basis to recognize the visual signs and act accordingly. We implemented our programming in a visual coding environment called Choregraphe which enables programmers to create complex behaviors. When used in conjunction with the Python programming language, Choregraphe can make projects more customizable [4]. Additionally, we utilized the Microsoft Kinect, along with the Microsoft Kinect C++ Software Development Kit (SDK), to record human body motions for the game prototype. After the collection and identification of body gestures, visual interpretation of the gestures were sent to the robot. In using this approach we aim to enhance human-robot interaction in order to solve real-world human-robot communication challenges.

This paper is organized as follows: Section 2 describes the background of the computational devices used. Section 3 provides the implementation of the project and demonstration. Section 4 introduces the discussion for real-world applications. Section 5 includes the conclusion and future work.

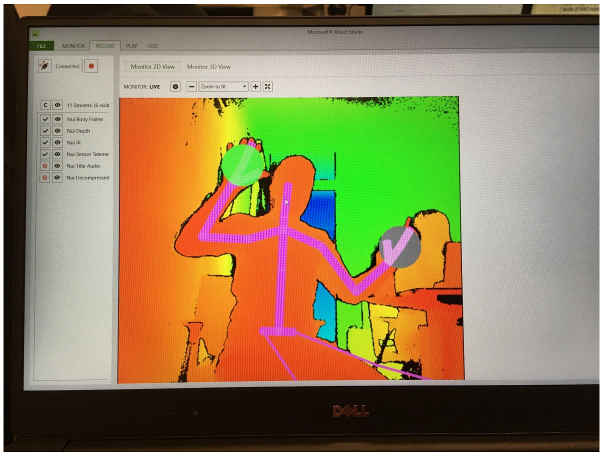
2. BACKGROUND

There are two devices used in this system, the NAO robot and Microsoft Kinect. The NAO robot is a programmable humanoid robot developed by Aldebaran Robotics, a French robotics company. The NAO robot is made up of a multitude of sensors, motors, and software piloted by a made-to-measure operating system: NAOqi OS [6] (Figure. 1).

<table>
<tr>
<td>



</td>
<td>



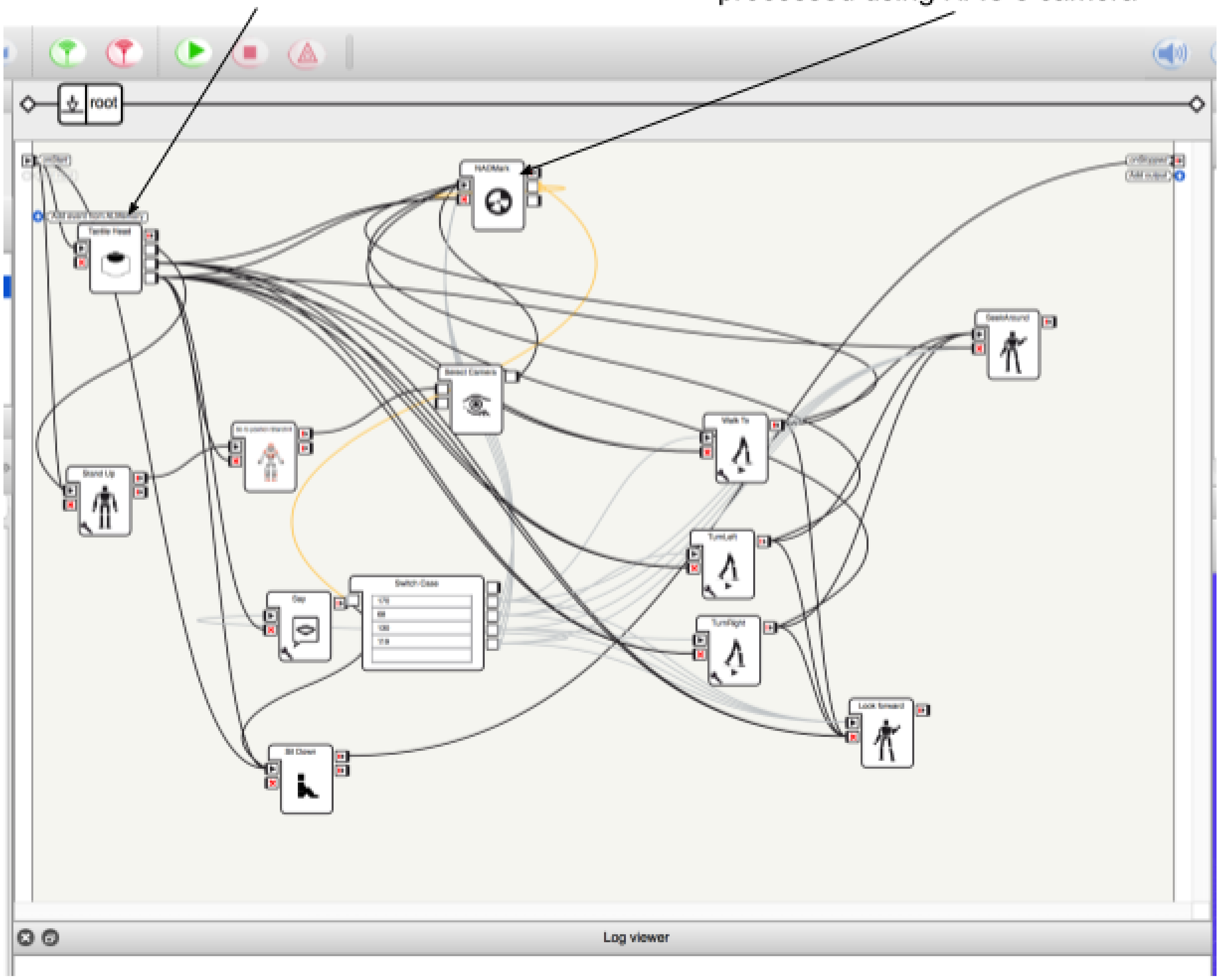
</td>
</tr>
<tr>
<td>Figure 1. Nao Robot and its sensors</td>
<td>Figure 2. Microsoft Kinect and its Skeletal Tracking</td>
</tr>
</table>

In additional to the NAO robot, we used Microsoft Kinect, a line of motion sensing input device produced for Xbox 360 and PCs. Microsoft Kinect has many features, such as Full HD color video and Skeletal Tracking [7] [9]. For this project, we mainly used the Skeletal Tracking features of Microsoft Kinect in order to track a person's gestures and motions (Figure 2). We then sent this data to NAO, allowing the robot to mimic the human's actions.

3. DEMONSTRATION

There are many approaches to solving real-world human-robot communication challenges using NAO, such as controlling the robot through voice commands or visual cues or gestures. We chose the maze challenge as an experimental environment to explore different approaches to human-robot communication. We took two different approaches for the NAO robot: (1) prompting the robot to follow human-provided visual cues to solve the maze game and (2) supplying the robot with human gestures recorded by the Kinect. This is an advanced human-robot interaction which allows humans to control the robot.

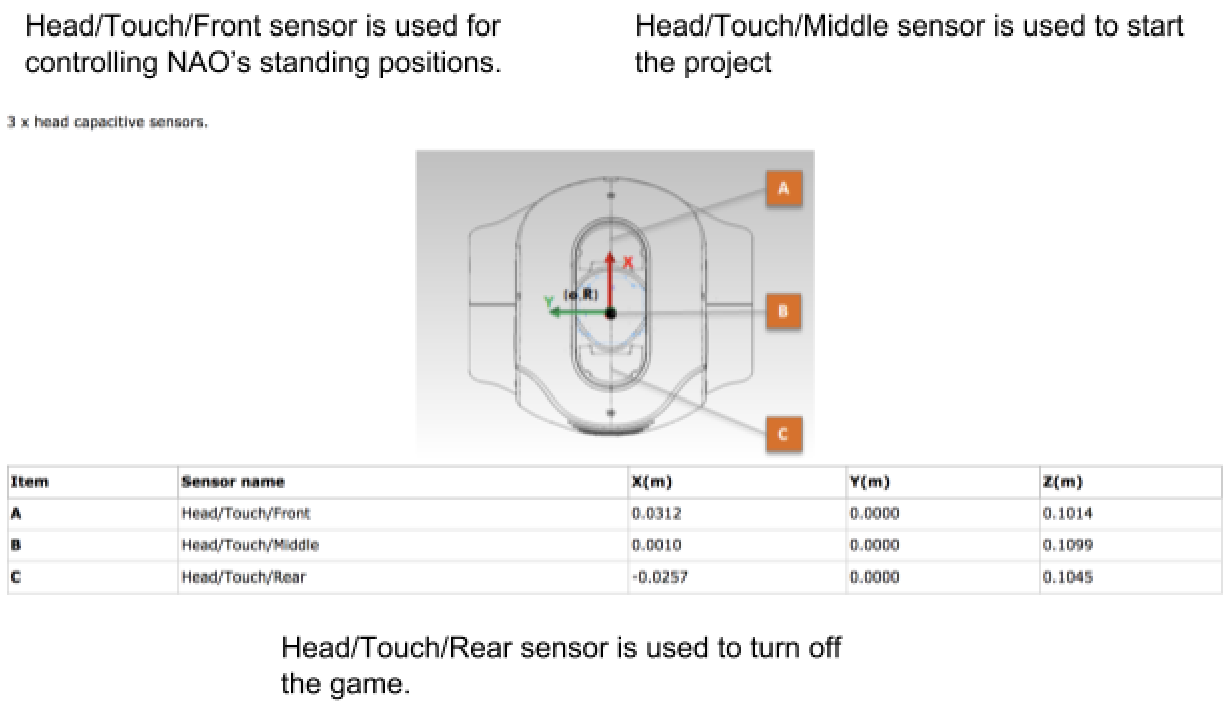
As seen in Figure 10, the first part of the maze game prototype we were able to fully implement was created in Choregraphe using drag-drop boxes and Python code.



head tactile of NAO robot

NAOMark visual cues were processed using NAO's camera

Figure 3: the project in Choregraphe



Head/Touch/Front sensor is used for controlling NAO's standing positions.

Head/Touch/Middle sensor is used to start the project

3 x head capacitive sensors.

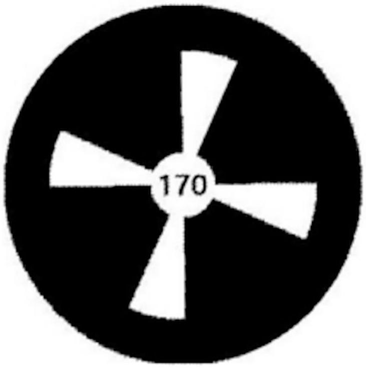
Item	Sensor name	X(m)	Y(m)	Z(m)
A	Head/Touch/Front	0.0312	0.0000	0.1014
B	Head/Touch/Middle	0.0010	0.0000	0.1099
C	Head/Touch/Rear	-0.0257	0.0000	0.1045

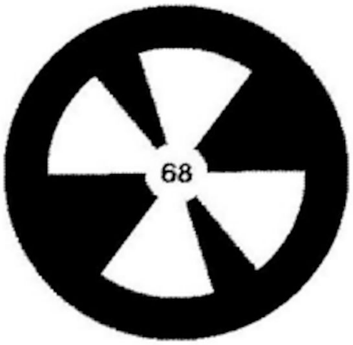
Head/Touch/Rear sensor is used to turn off the game.

Figure 4: Three head capacitive sensors of NAO

We programmed the NAO to begin the maze game in response to a human-provided touch to its Head/Touch/Front tactile sensor, one of the three head capacitive sensors located on the robot (Figure 4). After routine startup adjustments to ensure that the robot is physically in startup position, the NAO starts searching for NAOqi landmarks which are included in the vision module, ALLandMarkDetection [5] (Figure 5). The pre-defined NAOMarks within NAOqi are used as signs for the NAO robot to follow in order to finish the maze game. Human places the NaoMarks in locations where the NAO robot has to take some actions. For example, if the NAO robot sees a 130 NAOMark, the robot will turn left (Figure 5). In using a maze game, we aimed to create an environment in which the robot could be utilized to solve a real-world application through human-robot interaction.

Visual Commands for the Robot	Description of the command	Python Code

	Robot receives instructions to move forward	self.motion.setWalkArmsEnabled(self.getParameter("Left arm enabled"), self.getParameter("Right arm enabled"))
		
		self.motion.setMotionConfig([["ENABLE_FOOT_CONTACT_PROTECTION",self.getParameter("Stop walk when foot contact is lost")]])
		        for k, v in self.dFootGaitConfig.iteritems():
		            for unit in ["m", "rad", "%"]:
		                try: # try for each unit and do nothing if value in gaitConfig but not in parameters
		                    param = self.getParameter(str(k) + " (" + unit + ")")
		                    if(unit == "%"):
		                        param = param / 100.
		                    if(len(v) == 2):
		                        v.append(param)
		                    else:
		                        v[2] = param
		                    break
		                except:
		                    pass
		        gaitConfig = []
		        for k, v in self.dFootGaitConfig.iteritems():
		            try:
		                gaitConfig.append([k, v[2]])
		            except: # if some value added in gaitConfig but not in parameters
		                pass
		        # active walk process
		        self.motion.walkTo(self.getParameter("Distance X (m)"), self.getParameter("Distance Y (m)"), self.getParameter("Theta (rad)"), gaitConfig)
		         # The walk is finished so output

	Robot receives instructions to turn right	self.motion.setWalkArmsEnabled(self.getParameter("Left arm enabled"), self.getParameter("Right arm enabled"))
		
		self.motion.setMotionConfig([["ENABLE_FOOT_CONTACT_PROTECTION",self.getParameter("Stop walk when foot contact is lost")]])
		        for k, v in self.dFootGaitConfig.iteritems():
		            for unit in ["m", "rad", "%"]:
		                try: # try for each unit and do nothing if value in gaitConfig but not in parameters
		                    param = self.getParameter(str(k) + " (" + unit + ")")
		                    if(unit == "%"):
		                        param = param / 100.
		                    if(len(v) == 2):
		                        v.append(param)
		                    else:
		                        v[2] = param
		                    break
		                except:
		                    pass
		        gaitConfig = []
		        for k, v in self.dFootGaitConfig.iteritems():
		            try:
		                gaitConfig.append([k, v[2]])

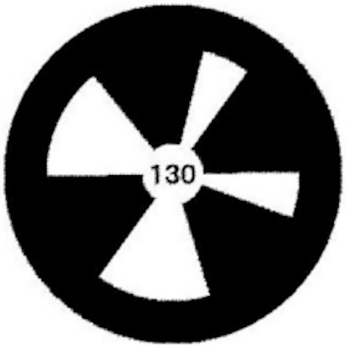
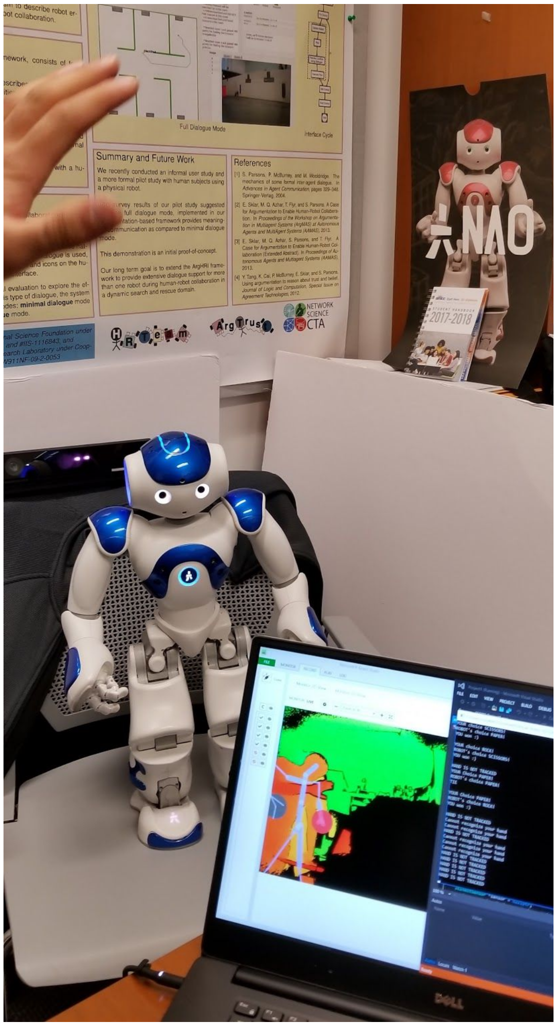
	Robot receives instructions to turn left	`except: # if some value added in gaitConfig but not in parameters`
		`pass`
		`# active walk process`
		
		`self.motion.walkTo(self.getParameter("Distance X (m)"), self.getParameter("Distance Y (m)"), self.getParameter("Theta (rad)"), gaitConfig)`
		`# The walk is finished so output`

Figure 5. NaoMarks and its usage in the project

The second part of the maze game uses Microsoft Kinect. This part has to be coded in C++ because Microsoft Kinect SDK requires the use of the C++ development environment. We divided the Kinect-NAO Robot communication challenge into two parts. First, we wanted to write a game prototype where a human plays a game with the computer using human gestures through Microsoft Kinect (Figure 6). Then we will work towards developing necessary software to connect Microsoft Kinect with NAO Robot. Our ultimate goal is to communicate with the NAO Robot using gestures through Microsoft Kinect.

The Kinect-Nao and human body gesture.



C++ Code that enables Microsoft Kinect to track body gestures of a human to play a game with computer.

```
void processBodies(const unsigned int & bodyCount, IBody * * bodies) {
    int iSecret;
    int count = 0;
    /* generate secret number between 1 and 3: */
    iSecret = rand() % 3 + 1;
    for (unsigned int bodyIndex = 0; bodyIndex < bodyCount; bodyIndex++)
{
        IBody * body = bodies[bodyIndex];

       //Get the tracking status for the body, if it's not tracked we'll
skip it
        BOOLEAN isTracked = false;
        HRESULT hr = body - > get_IsTracked( & isTracked);
        if (FAILED(hr) || isTracked == false) {
            continue;
        }

       //If we're here the body is tracked so lets get the joint
properties for this skeleton
        Joint joints[JointType_Count];
        hr = body - > GetJoints(_countof(joints), joints);
        if (SUCCEEDED(hr)) {
           //Let's print the head's position
            const CameraSpacePoint & headPos =
joints[JointType_Head].Position;
            const CameraSpacePoint & leftHandPos =
joints[JointType_HandLeft].Position;

           //Let's check if the use has his hand up
            if (leftHandPos.Y >= headPos.Y) {
                std::cout << "LEFT HAND UP!!\n";
            }

            HandState leftHandState;
            hr = body - > get_HandLeftState( & leftHandState);
            if (SUCCEEDED(hr)) {
                /*while (count != 3){
                                                       count++;
                                                       std::cout <<
count << "!" << std::endl;
                                                       wait(1);
                                            }
                                            wait(1);*/
                if (leftHandState == HandState_Closed && !isRock) {
                    std::cout << "YOUR choice ROCK!\n";
                    checkWhoWon(1, iSecret);
                    wait(2);
                } else if (leftHandState == HandState_Open && !isPaper)
{
                    std::cout << "YOUR Choice PAPER!\n";
                    checkWhoWon(2, iSecret);
                    wait(2);
                } else if (leftHandState == HandState_Lasso &&
!isScissor) {
                    std::cout << "YOUR choice SCISSORS!\n";
                    checkWhoWon(3, iSecret);
                    wait(2);
                } else if (leftHandState == HandState_NotTracked) {
                    std::cout << "HAND IS NOT TRACKED\n";
                    wait(1);
                } else if (leftHandState == HandState_Unknown) {
                    std::cout << "HANDS STATE IS UNKNOWN\n";
                    wait(1);
                }

            }
        }
    }
}

void checkWhoWon(int iUser, int iSecret) {
    switch (iSecret) {
        case 1:
            std::cout << "ROBOT's choice ROCK!\n";
            break;
        case 2:
            std::cout << "ROBOT's choice PAPER!\n";
            break;
        case 3:
            std::cout << "ROBOT's choice SCISSORS!\n";
            break;
    }
   // 1 is rock, 2 paper and 3 sciessors
    if (iUser == iSecret) {
        std::cout << "TIE \n";
    } else if (iSecret == 1) {
        if (iUser == 3)
            std::cout << "ROBOT won :) \n";
        else
            std::cout << "YOU won :) \n";
    } else if (iSecret == 2) {
        if (iUser == 1)
            std::cout << "ROBOT won:) \n";
        else
            std::cout << "YOU won :) \n";
    } else if (iSecret == 3) {
        if (iUser == 2)
            std::cout << "ROBOT won :) \n";
        else
            std::cout << "YOU won :) \n";
    }

    std::cout << std::endl;
}

void wait(int second_wait) {
    sleep_for(nanoseconds(10));
    sleep_until(system_clock::now() + seconds(second_wait));
}
```

4. Discussion: Real-World Application

Many of current approaches to human-robot communication are focused on verbal communication, however there are limitations when applied to real-world challenges such as problems understanding an accent or isolating external noise. Human-robot communication using gestures offers an alternative or supplement to verbal communication in cases in which verbal communication is limited. This is important in the search-and-rescue domain which may involve searching for victims, communicating with victims, and finding a safe path to victims for first responders [8]. The approaches introduced in this paper which solve a maze game prototype can be applied to human-robot communication for the search-and-rescue domain as well as many other aspects of our daily life.

Another way we can use approaches used in this paper is to support people with limited mobility or elderly people with physical limitations to walk. They would be able to control the robot to do their daily chores using either voice command or visual clues or body gesture or combination of the three.

5. CONCLUSION & FUTURE WORK

In this paper, we applied two different approaches to solve the maze game prototype. Our first approach involved using visual signs that the NAO robot can track and use to finish the maze game prototype. Our second, a more challenging and dynamic approach, used Microsoft Kinect motion tracking technology, and was implemented to solve the maze game prototype. By using these approaches, robots can collaborate with humans without a human physically being in an environment that can be overwhelming and dangerous. Our future work will include controlling the robot with human motions, thereby enhancing the interaction between human and robot.

References:

[1] Almetwally, I., & Mallem, M. (2013, April). Real-time tele-operation and tele-walking of humanoid Robot Nao using Kinect Depth Camera. In *Networking, Sensing and Control (ICNSC), 2013 10th IEEE International Conference on* (pp. 463-466). IEEE. https://ieeexplore.ieee.org/abstract/document/6548783/
[2] Cheng, L., Sun, Q., Su, H., Cong, Y., & Zhao, S. (2012, May). Design and implementation of human-robot interactive demonstration system based on Kinect. In *Control and Decision Conference (CCDC), 2012 24th Chinese* (pp. 971-975). IEEE. https://ieeexplore.ieee.org/abstract/document/6242992/
[3] Majgaard, G., & Brogaard Bertel, L. (2014, March). Initial phases of design-based research into the educational potentials of NAO-robots. In *Proceedings of the 2014 ACM/IEEE international conference on Human-robot interaction* (pp. 238-239). ACM. https://dl.acm.org/citation.cfm?id=2563690
[4] Choregraphe overview. *Aldebaran.* *http://doc.aldebaran.com/1-14/software/choregraphe/choregraphe_overview.html*
[5] ALLandMarkDetection. *Aldebaran*. http://doc.aldebaran.com/2-1/naoqi/vision/allandmarkdetection.html
[6] Find out more about NAO, *softbankrobotics.* *https://www.ald.softbankrobotics.com/en/robots/nao/find-out-more-about-nao*
[7] Skeletal Tracking, *Microsoft.* *https://msdn.microsoft.com/en-us/library/hh973074.aspx*
[8] Azhar, M. Q., Parsons, S., & Sklar, E. (2013, May). An Argumentation-based dialogue system for human-robot collaboration. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*(pp. 1353-1354). International Foundation for Autonomous Agents and Multiagent Systems. https://dl.acm.org/citation.cfm?id=2485221
[9] Skeletal Tracking, *Microsoft.* https://docs.microsoft.com/en-us/previous-versions/windows/kinect/dn782025(v%3dieb.10)