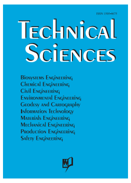
Technical Sciences, 2016, 19(1), 71–81

UNIWERSYTET WARMIŃSKO-MAZURSKI W OLSZTYNIE

Quarterly peer-reviewed scientific journal
ISSN 1505-4675
e-ISSN 2083-4527
TECHNICAL SCIENCES
Homepage: www.uwm.edu.pl/techsci/



CONTROL OF MINDSTORMS NXT ROBOT USING XTION PRO CAMERA SKELETAL TRACKING

Łukasz Żmudziński[1]***, Adam Augustyniak***[1]***, Piotr Artiemjew***[2]

[1] Robotic Circle
University of Warmia and Mazury in Olsztyn
[2] Department of Mathematics and Computer Science
University of Warmia and Mazury in Olsztyn

Received 28 July 2015; accepted 1 December 2015; available online 7 December 2015.

Key words: skeletal tracking, robot control, multi-user control, openGL.

Abstract

This paper focuses on the topic of creating a gesture-oriented user control system for robotic agents. For testing purposes we have used a Lego Mindstorm NXT self-designed robot and a Xtion Pro camera. Our construction consists of a NXT brick and three motors (two used for movement and one for the clutch mechanism). Our software is developed using C++ with NXT++ and OpenNi/NiTe libraries. System tests were performed in a real environment.

Introduction

In this work we present our gesture-oriented robot control system using C++ and the opensource libraries: NXT++ and OpenNi/NiTe. We explain the process of gathering user skeletal data (*OpenNi library for C++*. 2014) and transferring it into robotic movement (WALKER et al. 2014). The project was made as a part of (*OpenNi library for C++*. 2014). Lego Mindstorm acts only as a transmitter and all the calculations are done on a different machine.

The paper is divided in the following sections: section 2 presents information on the used libraries, section 3 describes the algorithm, section 4 shows

Correspondence: Piotr Artiemjew, Katedra Metod Matematycznych Informatyki, Uniwersytet Wrmińsko-Mazurski, ul. Słoneczna 54, 10-710 Olsztyn, phone: 89 524 60 82, e-mail: artem@matman.uwm.edu.pl

real life usage of the system and section 5 concludes the paper and shows future possibilities. Now we will show you, what are the reasons that motivated us in creating the project.

Motivation

Our main goal in creating the system is allowing the user to perform robotic movement with simple gestures; alone or in a team. We wanted to gather data on how well it performs compared to other steering methods and what possible use such a system could have in real world. Apart from that this is the first use of the connection between the provided libraries and making them work together. With that in mind, we would like to present some similar works by other authors.

Related Works

There are a few papers on the subject, but some of them especially caught our interest. The first one shows a mobile robot with Kinect on-board that tracks human movement and tries to keep the user in view of the camera (MACHIDA et al. 2012). Next paper (KOENEMANN, BENNEWITZ 2012) tracks human movement with a kinect-like device and transfers the data so that the robot agent mimics the moves. Third system focuses on usage of hidden Markov models (HMMs) to spot and recognize gestures captured with a data glove (IBA et al. 1999).

Library information

In this section we will show short information about the core libraries we used in our project – NXT++ (WALKER et al. 2014) and OpenNi/NiTe (*OpenNi library for C++*. 2014).

NXT++

In our project we use NXT++ for the sole purpose of accessing and steering our robotic agent. The library contains methods, that allow connection to the brick and motor movement invoked as follows:
|| NXT: :OpenBT(&comm)

Opens the bluetooth communication with the Lego NXT brick.

```
|| NXT: : Close(&comm);
```

Closes bluetooth communication.

```
|| NXT: : Motor : : Stop(&comm, port, brake);
```

Stops the motor at a specified port and sets, if the motor should use brakes.

```
|| NXT: : Motor : : SetForward(&comm, port, speed);
```

Sets the forward movement speed of a motor at a specified port.

```
|| NXT: : Motor : : SetReverse(&comm, port, speed);
```

Sets reverse movement speed of a motor at a specified port.

OpenNi/NiTe

OpenNi is a framework for natural interfaces (human-machine interface that is invisible to the user, who continuously learns increasingly complex interactions). NiTe is a module for OpenNi which focuses on seeing gestures and user limb movement.

The NiTe module allows us to assign each limb that we have interest in with a single method and then read the position values in three-dimensional space. Here is an example taken from our project that manages the right hand skeletal point movement of the application user:

```
joints . insert
(
pair<string , const nite : : SkeletonJoint>
(
"right hand" , user . getSkeleton (). getJoint ( nite : :JOINT RIGHT HAND)
)
);
if ( joints [ "righthand" ] . getPositionConfidence () > .5)
{ positions [ "righthand" ] [ ;x ; ] = joints [ "righthand" ] . getPosition (). x;
positions [ "righthand" ] [ 'y '] = joints [ "righthand" ] . getPosition (). y;
positions [ "righthand" ] [ 'z '] = joints [ "righthand" ] . getPosition (). z ; }
```

Firstly we add the right hand of the tracked user to the*joints* table and call it "right-hand". Then we read the x, y and z values of the joint (only if the confidence is higher than 50%, lower values can lead to corrupted results) and assign it to the added pair. This segment is looped in time so that we always get current position values.

Algorithm

Before we start going deep into the algorithm, we show the pseudocode below, which explains roughly how the whole process happens. We can divide it in three parts: initialization, data gathering and sending output commands depending on user selection and gestures. If the connection fails, the user is notified about the problem and the application stops.

```
IF connection to NXT established AND camera working
    User selects steering method
REPEAT ( Infinite )
Read camera data
    Find user skeleton
Assign joints
    Read three–dimensionaląpositions of joints
SWITCH steering mode
    IF joint positions in steering mode specified position
Send specific commands to NXT
ELSE
OUTPUT no connection message
```

The first part of the algorithm focuses on connections with the hardware and setting user preferences when it comes to steering methods, that he will be using. We start with checking the connection with the camera by executing the following command: ‖openni : : Status rc = openni : : OpenNI : : i n i t i a l i z e ();
If the return value is true, we proceed to establishing communication with our Mindstorm NXT robot. To achieve this, we need to call the OpenBT method as follows:

```
if (NXT: :OpenBT(&comm))
{
mindstrom _connection _open = true ;
} else
{ printf (''Can ' t_ _connect to
Mindstorm' );
return openni : :STATUS ERROR;
}
```

Just like before, if the value is true we continue to the user interface communication screen. In both cases if the value turns out to be false the program exits with an error message.

Now is the time that the user can select the steering method of his choice. He has three possible inputs:

1. Simple steering
2. Depth steering
3. Steering with clutches support

The methods vary in complexity of the gestures that the user can perform to steer the robot. *Simple steering* operates only within the X and Y axes. *Depth steering* and *Steering with clutches support* both add the Z axis, with the second of them using additional motors.

When the user accepts his choice, we proceed to the second part of the algorithm, that focuses on gathering limb information and processing the data for future purposes.

We start with gathering the camera data through the OpenNi/Nite library. To check, if we can continue we compare the value of the skeleton state, that the NiTe library returns after using the following method: users [i] . getSkeleton (). getState ();

The possible states that are returned are:

– SKELETON_NONE
– SKELETON_CALIBRATING
– SKELETON_TRACKED

Of course to continue with the gesture system, the skeleton must be tracked (without this, the algorithm will come to a stop).

Now we assign the joints, just like it was explained in section 2.2 of this paper. The whole skeleton representation in NiTe consists of 15 elements (Fig. 1), each of them having an unique name and that allows us to track all of them in a manner that fits our needs.

Once we get the position of each joint it is time to move onto the third and last part of our application – steering the robotic agent using NXT++.

Considering that we are connected to our NXT brick, the only thing that is left is to send specified commands. We achieve this by using the methods that were introduced in section 2.1. Of course to move the robotic agent we will only use the *SetForward*, *SetReverse* and *Stop* functions. Each of the steering methods has a different gesture system, that is why we will only show how the gestures are tracked and the complete steering guide will be explained in the next section (Testing).

As we already know, the joint positions are updated every frame, so the only thing that is left is to compare the values and execute specific NXT++ commands as follows:

```
if ( positions [ joint name a ] [ 'z '] >
positions [ joint name b ] [ 'z ' ] +
precisionX
&& positions [ joint name c ] [ 'y ' ] >
```

```
 positions [ joint name d ] [ ’y ’ ] +
precisionY )
{
NXT: : Motor : : SetForward(&comm, port, speed );
  NXT: : Motor : : SetForward(&comm, port,
    speed ); }
```

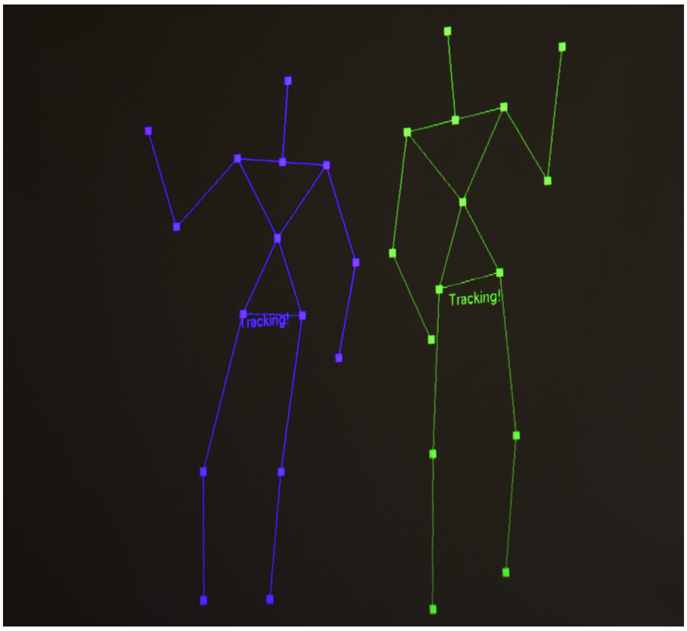


Fig. 1. Skeleton points

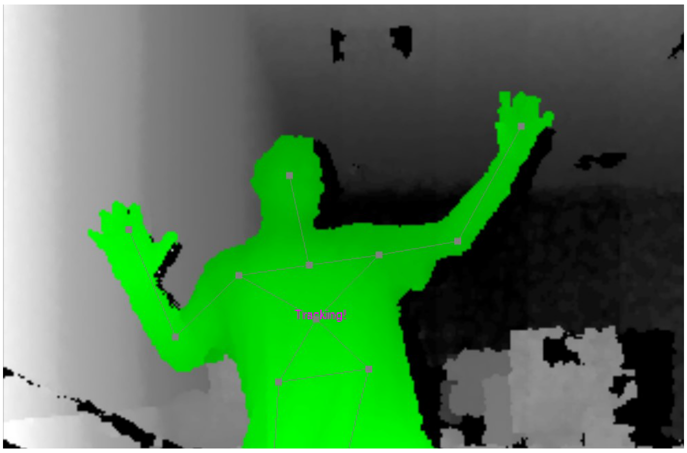


Fig. 2. Skeletal tracking as seen by user

As you can see, we introduced additional variables – precisionX and precisionY.

We use them to overcome the unwanted results with minimal movement of the users limbs. When none of the conditional statements are met, all the motors are stopped using the *Stop* method. The precision parameter was chosen in an experimental way, being highly dependent on the computer that was used. The parameter can be negative – but it's not recommended because we never encountered a situation where this would be needed.

When the user finished using the application, he can close the application by making a specific pose – crossing his arms on his chest for the time specified on the screen. It is explained in the following pseudocode:

```
 IF user is detected
GET user pose status
IF pose entered
OUTPUT Show countdown message
Start timer
ELSE IF pose exited
Output Show interrupt message
Stop timer
     ELSE IF pose is held
IF countdown completes
        Finalize connections
```

Working copy of actual algorithm code with instructions can be found on *NXT-XTION project* (2015).

Testing

Testing was an important part of the whole process, focusing on working aspects of the code in real life situations using actual hardware (ŻMUDZIŃSKI, AUGUSTYNIAK 2015). We divided this operation into the following phases:

– Camera connection and usage,

– NXT brick bluetooth connection, – Getting it all: steering methods.

Robot construction

In our tests we are using two robots – Black Claw (a robot built specifically for this project based on the Tribot model, *LEGO NXT Tribotąbuilding instructions*) and the Silver Shadow (ARTIEMJEW 2015). Our robot is constructed from various NXT specific parts. The main processor is located in the NXT Intelligent Brick 2.0 which features a powerful 32-bit microprocessor and Flash memory. It also supports bluetooth that allows us to establish a connection. The brick is linked with motors and sensors. Black Claw is using three engines – two for steering and movement and one for clutching. Apart from the input devices it is also using a LEGO Ultrasonic Sensor, which measures the distance to the closest object in front of the robot. All the parts are linked with connector cables with RJ12 plugs. The main frame is built from various LEGO parts.

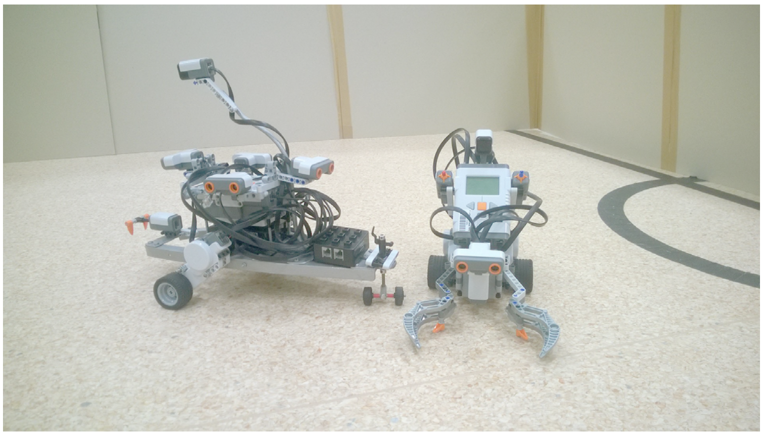


Fig. 3. Robots used in algorithm testing

Testing process

Camera connection and usage

First of all we started a self-written camera testing tool. The tool was written for the sole purpose of checking, if the camera works properly. When we finished installing all the drivers for XTion Pro camera we fired the mentioned application. We only encounter mediocre problems, that we corrected instantly within the code. After that we moved to the next step of our testing process.

NXT brick bluetooth connection

The main problem we met, was that the devices couldn;t find each other and then successfully pair (it was a hardware problem). After several tries and replacing bluetooth receivers, we finally managed to connect the robot with the main machine. After this, the connection was easily established and returned positive values in our code. Then we proceeded to testing the whole algorithm.

Getting it all: steering methods

To test the steering methods, we needed to run the application and get through all the previous steps. While testing, we got to the conclusion that the camera is not as accurate as we would want. Considering that fact, we had to add a precision parameter to all the gesture readings, to get the results within the minimal error margin. With three steering methods to pick, we had to test each of them respectfully. The testing resulted in making small amendments in all of them. Most changes were made to adjust the precision and the location collisions between joints. Moreover we modified the gesture system to make it easier for the user to transfer between different orders.

The final steering methods look as follows:

1. Simple steering
   - Forward movement
     Right hand directly above right shoulder (y-axis).
   - Reverse
     Left hand directly above left shoulder (y-axis).
   - Turn left
     Right hand above shoulder (y-axis) and left from right shoulder (x-axis).
   - Turn right
     Right hand above shoulder (y-axis) and right from right shoulder (x-axis).
2. Depth steering
   - Forward movement
     Right hand above shoulder (y-axis) and closer to the camera (z-axis).
   - Reverse
     Left hand above shoulder (y-axis) and closer to the camera (z-axis).
   - Turn left
     Left hand between shoulder and hip (y-axis), to the left of torso (x-axis).
   - Turn right
     Right hand between shoulder and hip (y-axis), to the right of torso (x-axis).
3. Steering with clutches support
   - Forward movement
     Right hand to the right of hip (x-axis).
   - Reverse
     Left hand to the left of hip (x-axis).
   - Turn left
     Left hand closer to the camera (z-axis).

– Turn right
  Right hand closer to the camera (z-axis).
– Open clutches
  Right hand directly above right shoulder (y-axis).
– Close clutches
  Left hand directly above left shoulder (y-axis).

The gesture system is constructed in the simplest possible way. Because of that, some gestures have a priority over others e.g. in the simple steering method lifting both hands over the shoulder will result in running the Forward movement method.

Additional testing

During the tests we checked how will the algorithm function when other users enter the scene as seen in Figure 4. Additional users are tracked and coloured respectfully in the process, but their movement doesn't affect the gesture system.

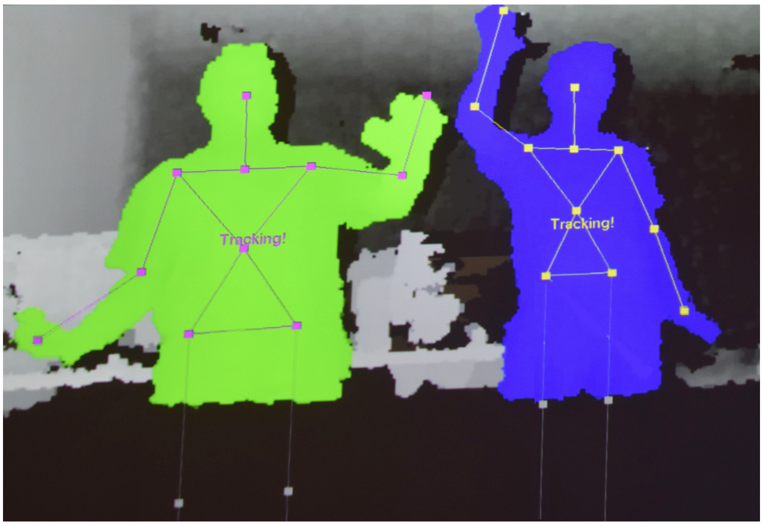


Fig. 4. Multi user test

Conclusions

The results of these experiments indicate that the combination of NXT++ with OpenNi/NiTE proved to be optimal. What is more its very easy and intuitive to add new kinds of steering methods. It comes from the fact that the whole process of assigning and reading joints and then transferring the gathered data through NXT++ to the agent is assimilable and fluid.

Focusing on understanding NiTE and OpenNI was the key to make the algorithm as simple as possible. This libraries enabled us to work freely with Xtion Pro camera and get the results fast from the very start.

Downloading data from Xtion Pro camera and skeletal tracking was necessary to give us the possibility to send various commands to the robot later. This part of the work is the most beneficial, because it creates a lot of future possibilities.

Thanks to the work of Piotr ARTIEMJEW (2015) we could use the NXT++ libraries which prove to be irreplaceable in making our robot controls. It was very comfortable to reconfigure data from the camera to create commands with NXT++.

Future works may develop more sophisticated usage of this kind of steering for example robot fights or delivering things without touching them. Of course gesture steering can be applied to other robots/machines like mechanical arms or humanoids (NAO) or even flying drones. Possibilities are nearly inexhaustible.

Acknowledgements

The research has been supported by grant 1309-802 from Ministry of Science and Higher Education of the Republic of Poland.

References

ARTIEMJEW P. 2015. *The localization of Mindstorms NXT in the magnetic unstable environment based on histogram filtering*. Conference: International Conference on Agents and Artificial Intelligence 2015 (ICAART2015), At Lisbon, Portugal, p. 341–348.

IBA S., WEGHE J.M.V., PAREDIS C.J.J., KHOSLA P.K. 1999. *An architecture for gesture-based control of mobile robots*. International Conference on "Intelligent Robots and Systems", IROS'99. Proceedings. IEEE/RSJ. Vol. 2.

KOENEMANN J., BENNEWITZ M. 2012. *Whole-body imitation of human motions with a Nao humanoid*. 7th ACM/IEEE International Conference on "Human-Robot Interaction" (HRI).

LEGO NXT Tribot building instructions. https://education.lego.com/dadk/lesi/support/product-support/mindstorms-education-nxt/nxt-base-set-9797/building-instructions.

MACHIDA E., CAO M., MURAO T., HASHIMOTO H. 2012. *Human motion tracking of mobile robot with Kinect 3D sensor*. Proceedings of SICE Annual Conference, Akita, p. 2207–2211.

NKR-UWM Robotic Circle of University of Warmia and Mazury. 2014. http://www.uwm.edu.pl/nkr/.

NXT-XTION project. 2015. https://github.com/nkruwm/nxt-xtion.

OpenNi library for C++. 2014. https://code.google.com/p/simple-openni/.

WALKER C., BUTTERWORTH D., ARTIEMJEW P. 2014. *NXT++ library*. https://github.com/cmwslw/nxt-plus-plus, http://wmii.uwm.edu.pl/~artem/downloads.html.

ŻMUDZIŃSKI Ł., AUGUSTYNIAK A. 2015. *Gesture system algorithm video presentation*. https://youtu.be/JppRyd3Vk1c.