Let’s Move IT – Unity as fleet manager

*Rene Bakx, April 2018*



Table of contents

1. Introduction 3

2. Playground setup 4

2.1. Playground overview 4

2.2. Playground components 5

2.3. Playground specifications 5

3. Playground experiment 6

4. Results 7

5. Conclusion 8

6. Bibliography 9

Appendix 1 EV3 to Unity API 10

Appendix 2 Vision to Unity API 11

Appendix 3 Investigation: connection of Unity to EV3 Mindstorms 12

# Introduction

In this document the work on Let’s Move IT [1] is described. This work is part of the Fontys FHICT activities on the two work packages: WP1 (planning and control) and WP4 (test area mobile robots).

The main research question to be answered:

* “*Can Unity be used as a fleet manager for simple mobile robots*?”

To answer this question we want to address the following subquestions:

* *“Can Unity control multiple physical robots?”*
* *“How useful are Unity tools like behavior trees, decision trees, pathfinding for fleet management?”*
* *How would Unity compare to existing fleet management systems?*

To answer these questions a playground is setup to enable students and teachers to do experiments with Unity as a fleet manager. These experiments will give insights and hopefully will give answer to the research questions.

# Playground setup

## Playground overview

See Figure 1 below for an overview of the setup.

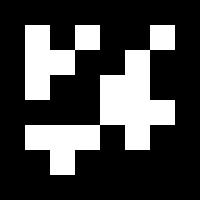
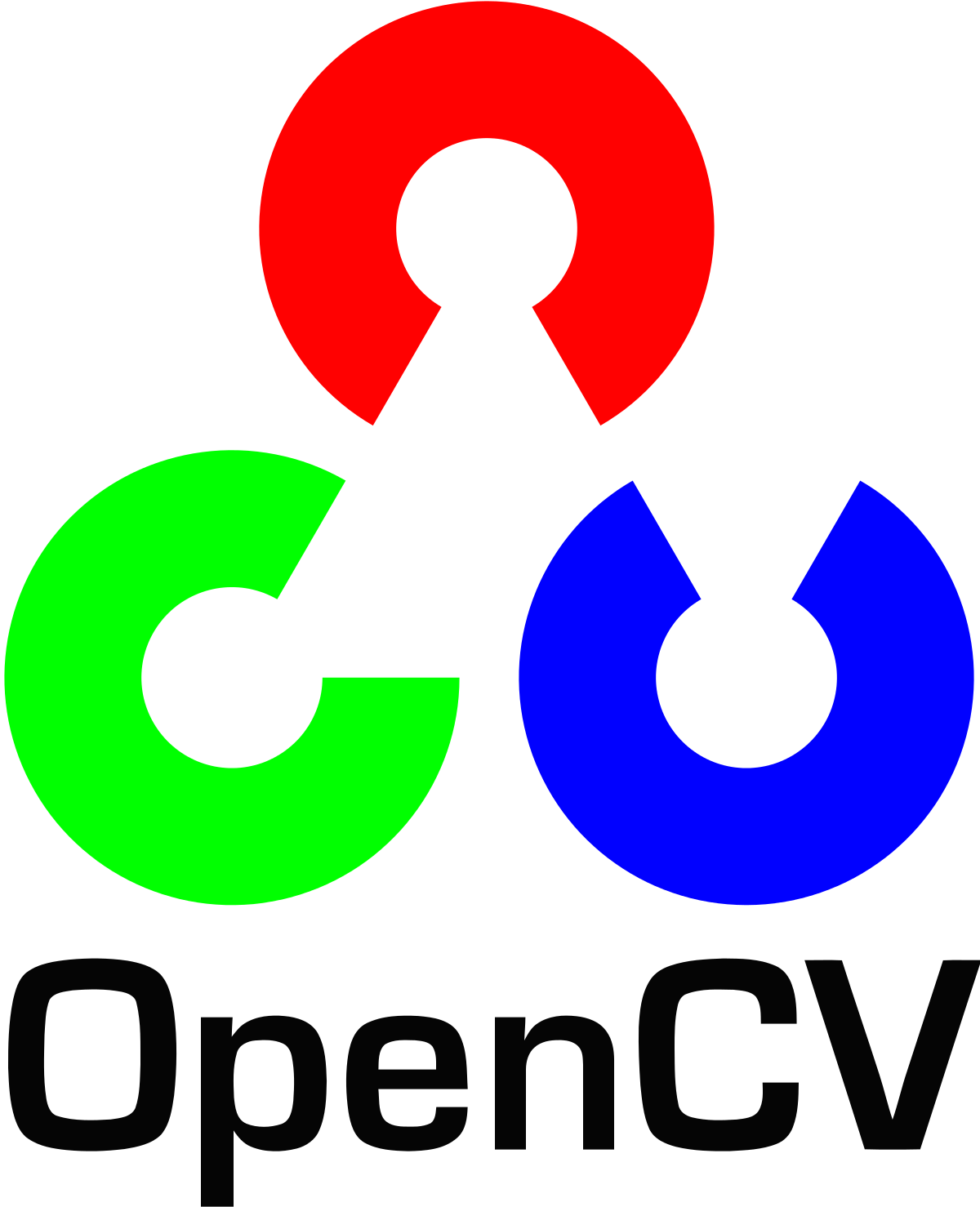


sensor data

move commands



laptop running Unity and openCV



position and orientation of markers



Beamer displaying the Unity scene on the Playground on the floor

Playground with EV3’s with ArUco marker



Camera overviewing the playground

Figure Playground setup

The use of a playground with simple robots like de Mindstorms EV3 enables FHICT Technology students to participate in a practical way in this research. The playground will be virtualized in a Unity scene. In Unity also the robots are virtualized and perform a task in the virtual world. The actions of the virtual robots are sent to the physical robots while sensor data of the physical robots is sent back to Unity. The sensor data includes the vision data obtained by a camera above the playground. This enables Unity to correctly place the virtual robots in the virtual world and calculate appropriate actions. To make the playground more vivid the Unity scene can be projected on the playground using a beamer.

## Playground components

At <https://github.com/rbakx/UnityRobot> the code for the project can be found. The main components are:

* EV3Unity project.  
  This project is the Unity project. When a connection with the physical EV3 is made, the EV3 will actually push a real ball into the goal. When there is no connection, a simulation minics the behavior of the EV3 in Unity.
* VisionServer written in Python. This server receives vision data from OpenCV about the location and orientation of the EV3 robot and the position of a ball if present. The vision information is sent to Unity in JSON format.
* EV3Unity.c.  
  This is a ROBOTC file which runs on the EV3. It executes move commands received from Unity and sends back sensor information of the motor encoders and the Gyro.
* EV3Wifi library.  
  This library is used by Unity to communicate to the EV3 robot. It is a C# library for socket communication with the EV3.

## Playground specifications

The specifications of the setup are as follows:

* MacBook Pro 2015 running both Unity 2017 and OpenCV 3.
* Camera: Logitech C930: resolution 1920x1080@30fps,, 90° field of view, mounting height 250cm.
* LEGO Mindstorms EV3: ROBOTC firmware version 1.07X.
* Aruco marker size: 8x8 cm.
* Beamer DLP Optoma.

# Playground experiment

For the experiment a Unity scene is created which consists of a soccer field in which a virtual robot and a ball are placed. See Figure 2.



Figure 2 Soccer experiment

When pressing ‘g’ the robot pushes the ball towards the goal and scores. When connected to a physical EV3, the EV3 will do the same. The Vision system provides location and orientation information of the EV3 and location information of the ball.  
This experiment was chosen because the ball movements are dynamic and unpredictable. When the feedback loop between Unity and the physical playground can deal with this scenario it should be able to deal with other useful scenarios.

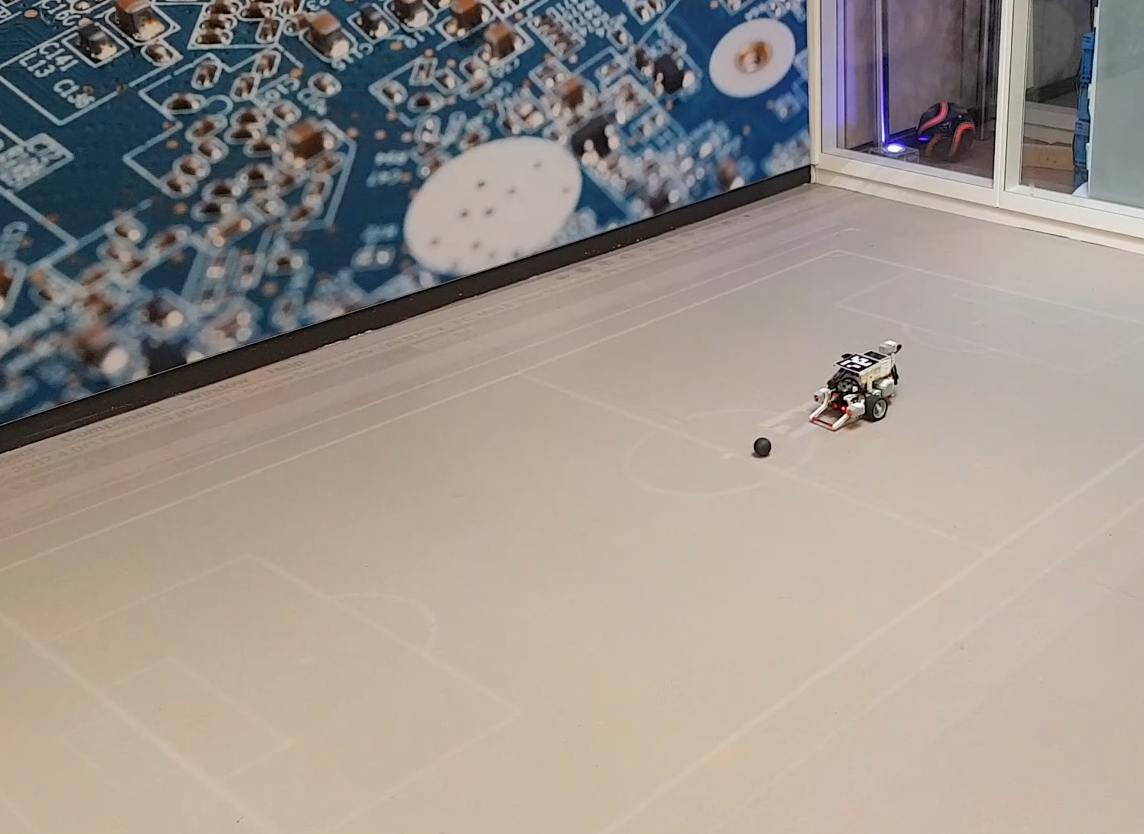
Below a picture of the actual playground.

Figure Actual playground

# Results

At <https://youtu.be/mdvvy58DPDM> a demo video of the soccer experiment is available. Below some concrete results are listed.

* Refresh rate in Unity: 10fps
* Positioning accuracy on the playground: 5 cm
* Out of 10 times, the EV3 scores 8 times. Because the EV3 moves slowly the ball is also pushed slowly. Because the floor is not flat the ball can roll sidewards and miss the goal. This can be solved by increasing the speed of the EV3 so the ball is pushed with more force. This requires more computing power to increase the framerate to keep up with the faster movements.

# Conclusion

From the results above we conclude Unity is capable of controlling a physical robot and that the tools provided by Unity can be used to give a physical robot a meaningful task. Future experiments have to be conducted to test how scalable the provided framework is in both performance and in number of robots. Only then the first sub question can be answered.

Possible future experiments are listed below. They are listed per sub question.

*“Can Unity control multiple physical robots?”*

* Adding more robots to the playground and let them interact.
* Adding different types of robots to the playground and let them interact.
* Define a robot API for performing meaningful tasks.
* Define a robot API for announcing its capabilities.
* Let a robot find and connect automatically to its charging station. This would enable a continuous demo.

*“How useful are Unity tools like behavior trees, decision trees, pathfinding for fleet management?”*

* Define some typical fleet management tasks and implement these with the Unity setup.

*“How would Unity compare to existing fleet management systems?”*

* Compare the Unity implementations with existing fleet management solutions.

# Bibliography

1. Kiela, H. (2017). RAAK MKB Project LET’S MOVE IT. Retrieved from <https://www.nwo.nl/onderzoek-en-resultaten/onderzoeksprojecten/i/89/29489.html>
2. Bakx, R. (n.d.). UnityRobot project. Retrieved May 17, 2016, from https://github.com/rbakx/UnityRobot
3. Bartneck, C., Soucy, M., Fleuret, K., & Sandoval, E. B. (2015). The Robot Engine - Making The Unity 3D Game Engine Work For HRI. *Proceedings of the IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN2015)*, *RO-MAN2015*, 431-437. doi:10.1109/ROMAN.2015.7333561
4. Singhal, A., Kejriwal, N., Pallav, P., Choudhury, S., Sinha, R., & Kumar, S. (2017, November 9). Managing a Fleet of Autonomous Mobile Robots (AMR) using Cloud Robotics Platform [IEEE Xplore]. Retrieved May 17, 2018, from https://ieeexplore.ieee.org/document/8098721/

# Appendix 1 EV3 to Unity API

using EV3WifiLib;

EV3WifiLib namespace

myEV3.Connect ("1234", ipAddress);

Connect to EV3. Returns true if successful, else false

myEV3.Disconnect ();

Disconnect from EV3. No return value.

myEV3.isConnected

true if EV3 is connected, false otherwise.

myEV3.SendMessage ("Move 0 10 30", "0");  
First parameter is a string in the format: “Move angle (-180 .. 180) distance (cm) power (0..100)”  
Second parameter is the EV3 inbox number which is “0”.

myEV3.ReceiveMessage ("EV3\_OUTBOX0");

Receives a string with 4 numbers separated by a space.

int taskReady:

indicates whether EV3 is ready for the next task or not, true or false

float angleMoved:

indicates the angle in degrees the EV3 has actually moved (using gyro sensor), -180..180

float distanceMoved:

indicates the distance in cm the EV3 has actually moved (using motor encoders), -inf, inf

float distanceToObject:

indicates the distance in cm to a detected object: 0..inf

# Appendix 2 Vision to Unity API

myVision.Connect ("1234", ipAddress);

Connect to vision. Returns true if successful, else false

myVision.Disconnect ();

Disconnect from vision. No return value.

myVision.isConnected

true if vision is connected, false otherwise.

public class VisionData  
{      
    public float[] videoSize = { 0.0f, 0.0f };  
    public float[] bot1 = { 0.0f, 0.0f, 0.0f };  
    public float[] ball = { 0.0f, 0.0f, 0.0f };  
}

Vision data is received in JSON format which currently can be deserialized into the above class. The bot1 data consists of three floating point numbers, in order:

1. Angle of the bot
2. Horizontal position of the bot
3. Vertical position of the bot

The ball data is the same, except the first parameter represents the size of the ball.

# Appendix 3 Investigation: connection of Unity to EV3 Mindstorms

Connecting the EV3 Mindstorms robot to a host like Unity is not trivial. Several options are investigated. A difficulty is that Unity 2017 4.0f1 supports .NET 3.5. This means that existing code or libraries using .NET > 2.0 (which most do) cannot be used within Unity.  
**At the Build conference 2016 Microsoft announced that Unity will join the .NET Foundation, so in the future it will be easier to use existing code and libraries.**

**Connecting using Bluetooth and EV3Messenger**

Making use of <https://ev3messenger.codeplex.com/> it is possible to connect a host to the EV3 using Bluetooth. However, some .NET 4 functionality is used: the queuing mechanism in System.Collections.Concurrent. So this has to be replaced. In Windows an applications connects to a Bluetooth device via a serial port emulation. On the Mac OS X MonoDevelop-Unity is used which does not support this, so it is not possible to use the EV3Messenger code with Unity on a Mac.

Advantage of using Bluetooth to connect to the EV3 is that mailbox messaging is supported for sending and receiving. Using Wifi, sendig to a EV3 mailbox is supported, receiving seems not to be supported.

**Connecting using Wifi and MonoBrick**

Making use of <http://www.monobrick.dk/> it is possible to connect a host to the EV3 using Wifi. For the EV3 only the 'NETGEAR WNA1100 - N150 Wireless USB Adapter' is supported. It is tested that with MonoBrick it is indeed possible to send mailbox messages to the EV3 using Direct Commands. The MonoBrick library uses .NET 4 functionality so will not work with Unity. Receiving messages seems not to be supported. However, with Direct Commands it is possible to retrieve the sensor data. Also it is possible to read memory. It still has to be investigated whether it is possible to write the same memory from a standard EV3 program. If this is the case, it is possible to send messages to an EV3 using a mailbox and receive back messages through memory.

**Connecting using Wifi and the LEGO MINDSTORMS EV3 API for .NET**

Instead of MonoBrick, one can make use of <https://github.com/BrianPeek/legoev3>. These libraries also make use of .NET 4 functionality so will not work with Unity. I tested this only using a USB connection which worked.

**IMPORTANT NOTE**: When building the TestApplication provided by MonoBrick it might be that when starting the error message “Error: Failed to open connection” appears. Resolution: set Platform target from ‘Any CPU’ to ‘x86’ and (if needed) copy hidapi.dll and hidapi.dylib from the LEGO Software installation to the Monobrick Test Apllication.

**Connecting using Wifi and a simple TCP/IP connection**The EV3 can connect throug Wifi. Only the 'NETGEAR WNA1100 - N150 Wireless USB Adapter' is supported. How connect a host to EV3 through Wifi is described at <http://www.monobrick.dk/guides/how-to-establish-a-wifi-connection-with-the-ev3-brick/>. In short:

* When the EV3 wifi is enabled and connection is made to the network, the EV3 starts broadcasting an UDP message every 10 seconds. This broadcast contains the serial number of the EV3.
* The host now knows the EV3 IP address and serial number and replies to this message to let the EV3 know it can expect a connection request.
* The host sends a TCP/IP connection request.
* The EV3 accepts the connection.
* The host sends an unlock message using the Serial number of the EV3.
* The host can now send commands to the EV3 using the TCP/IP connection.

The above connection sequency can easily be implemented in C# with .NET 3.5 functionality which is used in Unity.

**IMPORTANT NOTE**: When running the C# application on a VirtualBox virtual machine, be sure that the EV3 robot is on the same network as the virtual machine. This means that in VirtualBox ‘Devices -> Network -> Network Settings -> Attached’ must be on ‘Bridged Adapter’ and not on ‘NAT’.

**Sending messages from host to EV3**

After the connection is established System Commands or Direct Commands can be sent to the EV3. One of the commands is 'WRITEMAILBOX' which can be used to send a message to a receiving mailbox on the EV3.

**Sending messages from EV3 to host**

Sending back messages from the EV3 to the host through a mailbox seems only to be supported for Bluetooth, not for Wifi.

At <https://siouxnetontrack.wordpress.com/2014/08/19/sending-data-over-wifi-between-our-pc-application-and-the-ev3-part-1/> a workaround is decscribed. On the EV3 it is possible to write text to a file which can be read on the host using the opFile(READ\_VALUE) command (byte code command for the Virtual Machine) packed in a Direct Command. There also are System commands which are not handled by the VM.

Attention point with this method is that when the EV3 is writing the file, the host cannot read it. A way around this is to let the EV3 write a new value in the file after a request from the host (e.g. 'get\_distance') and then let the host close the file after which the host can read the value.

**EV3WifiLib library**

For sending and receiving messages EV3WifiLib.dll is created containing the EV3Wifi class. This library is compiled with 'Target framework' set to '.NET Framework3.5'. Next it can be included in a Unity project. In Unity, set the 'Edit -> Project Settings -> Player -> Api Compatibility Level' to '.NET 2.0'.