SYSTEM DESIGN PROJECT 2012

MILESTONE 1 INDIVIDUAL REPORT

MARTIN MARINOV S0932707(ALPHA TEAM – TEAM 8)

TEAM ALLOCATION

Initially I chose to be in Bluetooth communication team. I eventually also ended up writing parts of the code that is running on the Brick itself. Since we had to coordinate closely with the team that was physically building the brick, I was involved partially in taking design decisions.

After we finished the Brick and the onboard software, the communication team had to be reallocated. I started working on the AI and implemented the AI framework which would be used in future for taking decisions and implementing strategies.

BLUETOOTH

I made a *Communicator* interface which provides an abstraction layer for communication between different parts of the project. Currently there is a PC and a Brick *Communicator*s which implement the actual Bluetooth communication. Messages sent to and from the brick are of type [**opcode**] [**N**] [**arg1**] [**arg2**] … [**argN**] where every item between [and] is a byte. This is done for efficiency purposes and to minimize bandwidth. The *opcodes* are defined in a shared *enum* declared in the *Communicator* interface. Sending and receiving messages is done asynchronously using Java *Threads*. See ***Figure 1*** for a code sample that could be run on PC to send a command to the Brick to make it move for 2 seconds. I was also responsible for adapting the third party libraries to DICE and writing tutorials in the wiki to help my teammates set up their IDEs.

BRICK SOFTWARE

I took part in writing almost all of the commands. I am particularly proud with the operate command which would be the main interface between the AI and the Brick. It unifies all possible special movements into a single command. After extensive research on last years' projects, I discovered that they had a mess of commands which overcomplicated the logic of the projects. I proved my point by creating an interactive manual "joypad-style" control over robots' movements using only the operate command (see ***Figure 2***).

AI

Currently *AI* is in its early stages. I've implemented a low-pass filter which filters the incoming data from the visual system. This helps removing high-pitched noise from uncertainties in the visual system. I also started building some simple *AI* commands based on visual servoing. I also implemented a GUI for testing the proper functioning of the *AI* (see ***Figure 3***).

Appendix

```java
public class Test implements MessageListener {
    public Test(){
        // initialize bluetooth, registering the current class as a listener
        Communicator my = new JComm();
        my.registerListener(this);
        // send move message
        my.sendMessage(opcode.move, (byte)2);
        // wait until move is done
        Thread.sleep(5000);
        // after 5 seconds send an exit message
        my.sendMessage(opcode.exit);
    }

    @Override
    public void receiveMessage(opcode op, byte[] args, Communicator con) {
        // this will get triggered if the brick sends a message back
    }
}
```

Figure 1 – Illustrated usage of the Communicator interface I was responsible for

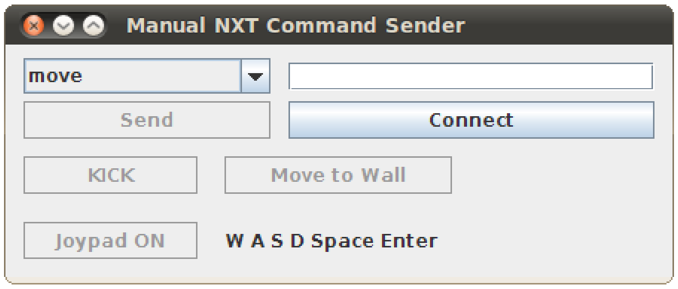
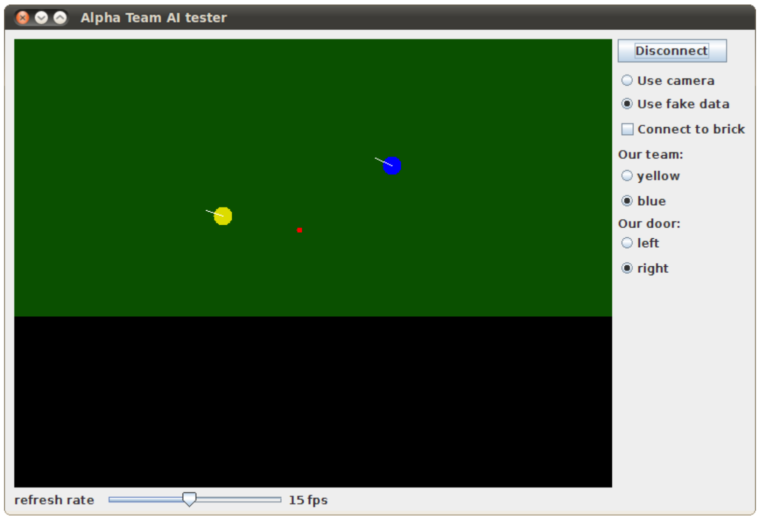


Figure 2 – Manually dispatching commands to Brick. If Joypad mode is on, buttons W, A, S, D could be used to control robot



Use camera
[✓ Use fake data]
Connect to brick
Our team:
yellow
[✓ blue]
Our door:
left
[✓ right]
refresh rate 15 fps

Figure 3 – GUI for testing AI. The image is generated by what the AI "sees" on the field.