IoT Frontends with Apache Flex

Christofer Dutz <christofer.dutz@codecentric.de>

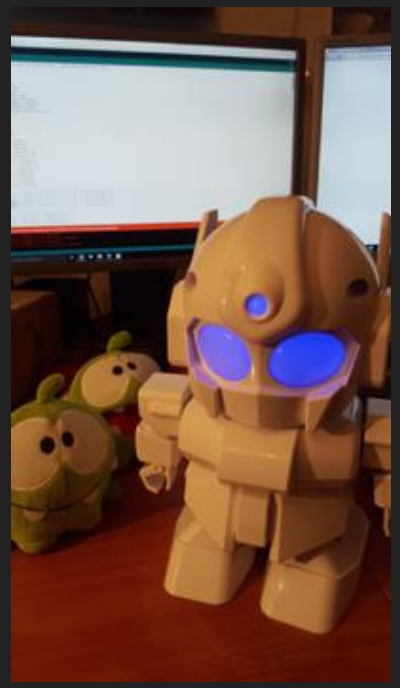
ApacheCon EU 2016 - Seville - 2016-11-16

ABOUT ME

- Christofer Dutz
- Senior IT Consultant
- codecentric AG
- Apache Member
- Apache Flex Committer & PMC
- Flexmojos Lead Developer
- Open-Source enthusiast

AGENDA

- What do we need?
- What parts of Apache Flex can help?
- Why is Flex so great for IoT?
- Demo



WHAT DO WE NEED?

- Server application a client can talk to
- Client that talks to the server
- Logic on the server that talks to the IoT hard/software

WHAT PARTS OF APACHE FLEX CAN HELP?

- BlazeDS as communication endpoint to any client
- Flex client as frontend

WHY IS FLEX SO GREAT FOR IOT?

ON THE SERVER

- BlazeDS-Spring-Boot-Starter
  - Automatically configures the BlazeDS server
  - Using Spring-Flex for configuring the details
  - Activated by providing a "META-INF/flex/services-config.xml"

ON THE SERVER

- BlazeDS
  - Server-Push
  - Remoting (call method on server)
  - Messaging (JMS-Like publish-subscribe)
    - JMS integration possible
  - Automatic serialization/deserialization
  - Strongly typed

ON THE SERVER

- Spring-Flex
  - Make Spring bean accessible by annotating with @RemotingDestination
  - Takes care of configuring BlazeDS for Spring
  - Integrates BlazeDS into Spring
    - Spring Messaging
    - Spring Security
    - ...

ON THE CLIENT

- Strongly typed model
- Client-Side model auto-generated from server model
- Framework handles the communication details automatically
- Deployable as native application on Android and IOs devices
- Skinning

CLIENT-SERVER COMMUNICATION EXAMPLE

ON THE SERVER

```
package de.codecentric.iot.rapiro.movement;

import de.codecentric.iot.rapiro.movement.model.MovementState;
import org.springframework.flex.remoting.RemotingDestination;
import org.springframework.stereotype.Service;

@Service("movementService")
@RemotingDestination
public class MovementService  {

    private MovementState movementState;

    public void stop() {
        // Do something
    }
```

ON THE CLIENT

- Declaration of the remote service incl. methods and callbacks

```
<s:RemoteObject id="movementService"
                destination="movementService"
                fault="onFault(event)">
    <s:method name="stop" result="onResult(event)"/>
    <s:method name="moveForward" result="onResult(event)"/>
    <s:method name="getMovementState" result="onGetMovementSt
</s:RemoteObject>
```

ON THE CLIENT

- Calling the remote methods

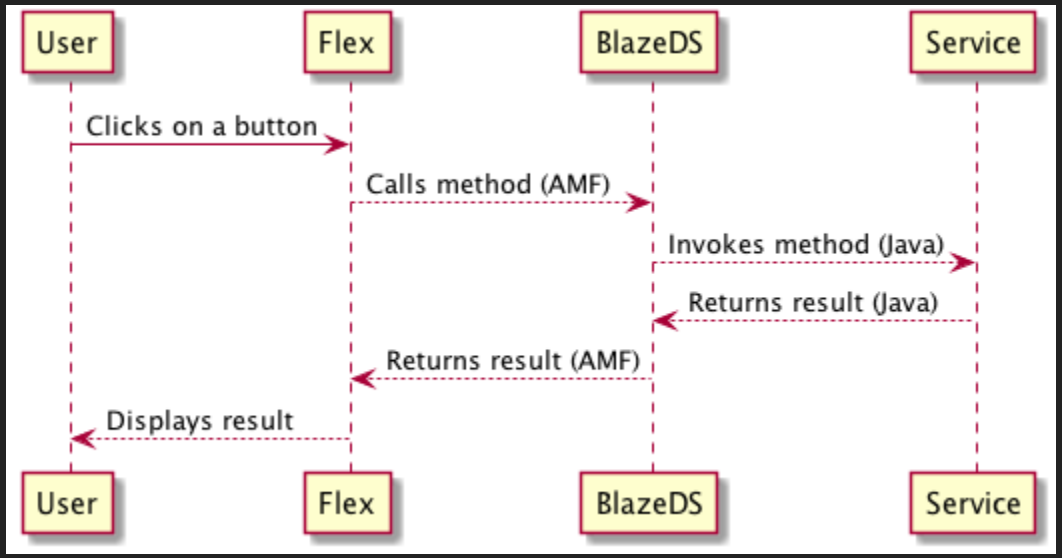
```
<s:Button click="onStopClick(event)" label="Stop"
          enabled="{enabled}"/>
```

```
protected function onStopClick(event:MouseEvent):void {
    movementService.stop();
}
```

```
protected function onResult(event:ResultEvent):void {
    // Handle the result.
}
```

```
protected function onFault(event:FaultEvent):void {
    // Handle the error.
}
```

BEHIND THE SCENES



SKINNING

- Component + Skin
- Component = functional implementation of control
- Skin = visual representation (incl. UI logic)
- Not only for positioning as with CSS in JS/HTML
- Different usability on touch and desktop

DEMO OF THE ROBOT

HARDWARE DETAILS

- Arduino Uno handling the Movement
- Pixy (CMUcam5) handling processing of the image data
- IR Distance sensor
- Raspberry PI 3 communicating with all of the sub-systems
  - 64 bit Quad Core ARM Cortex-A53
  - WiFi access point, 32GB SD flash memory, 1GB Ram
- PAM8403 based audio amplifier

SOFTWARE DETAILS

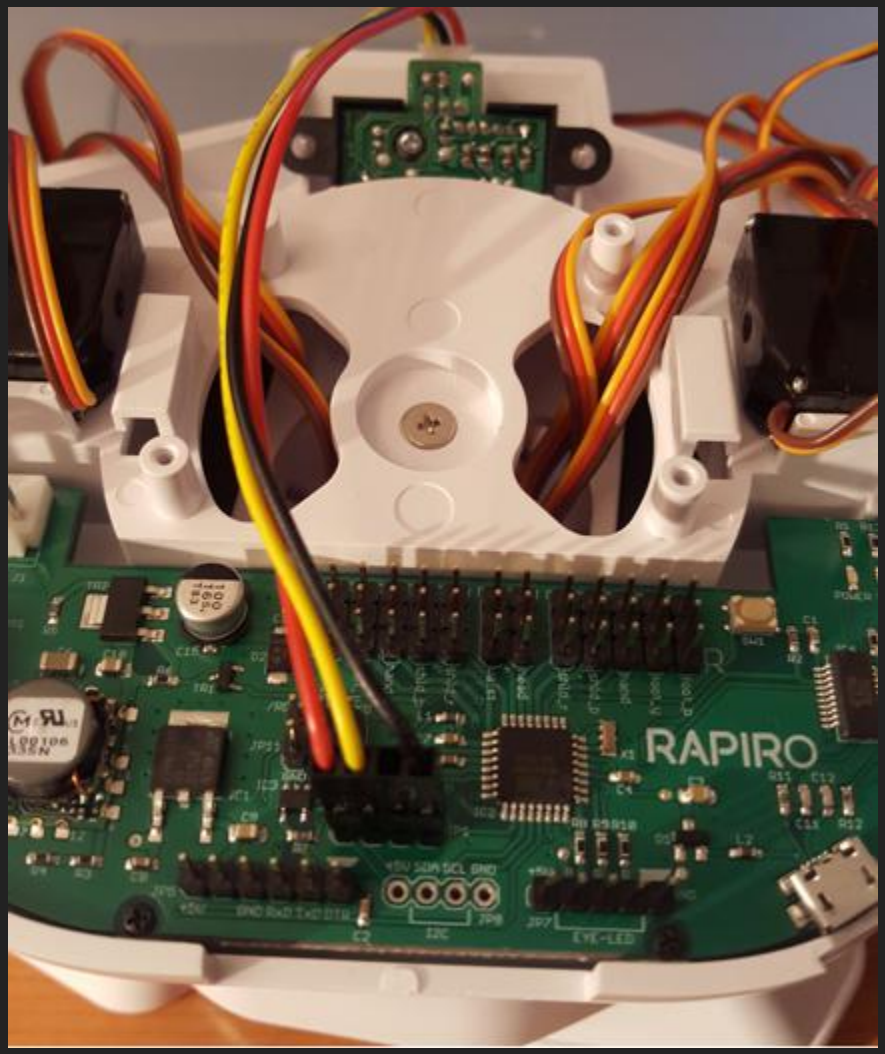
- OS: Raspbian
- Oracle Java 1.8 JDK
- Scala 2.11.8
- Jetty-based Spring-Boot application
  - Spring 4.3.2.RELEASE
  - Spring-Boot 1.4.1.RELEASE
  - Spring-Flex 1.5.2.RELEASE
  - Spring-Scala 1.0.0.M2

SOFTWARE DETAILS (CONTINUED)

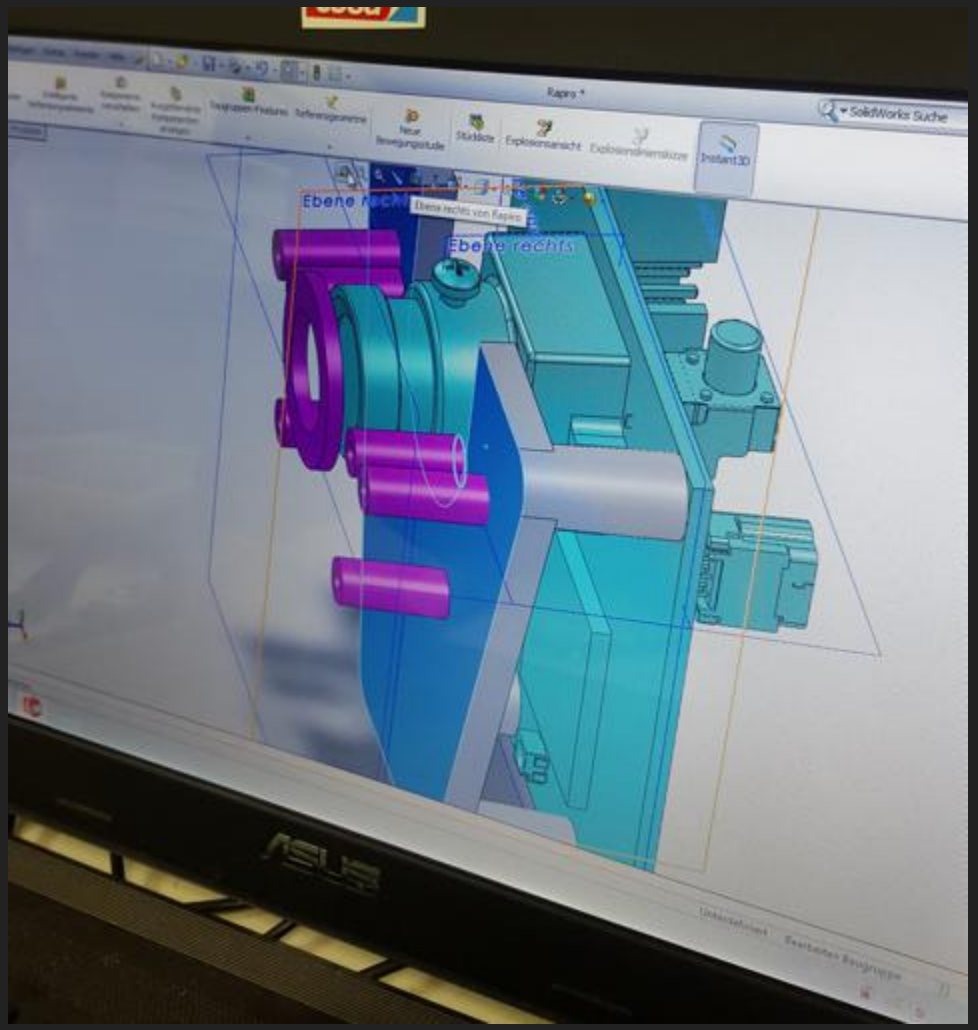
- PI4J 1.1 (lib for accessing RP devices)
- BlazeDS 4.7.3-SNAPSHOT (for client communication)
- Akka 2.4.11 (for higher "brain" functions)
- Sources available on Github at: https://github.com/chrisdutz/RAPIRO

IMPRESSIONS

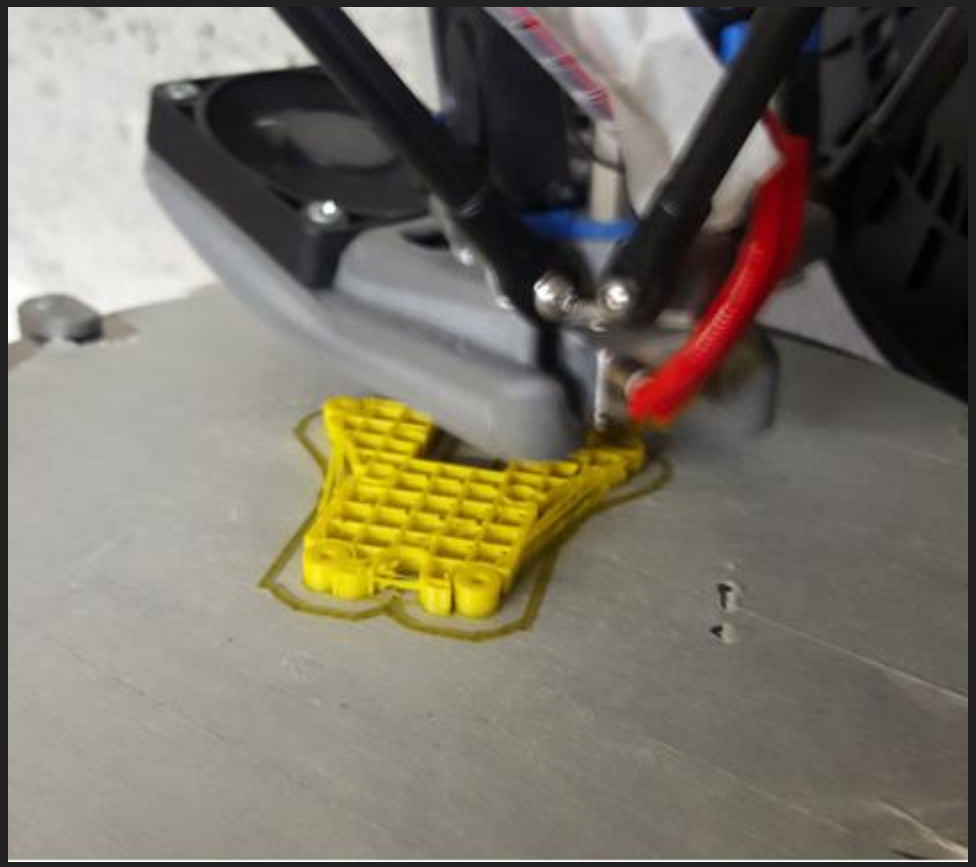
ARDUINO



CAD DESIGNING THE CAM CARRIER

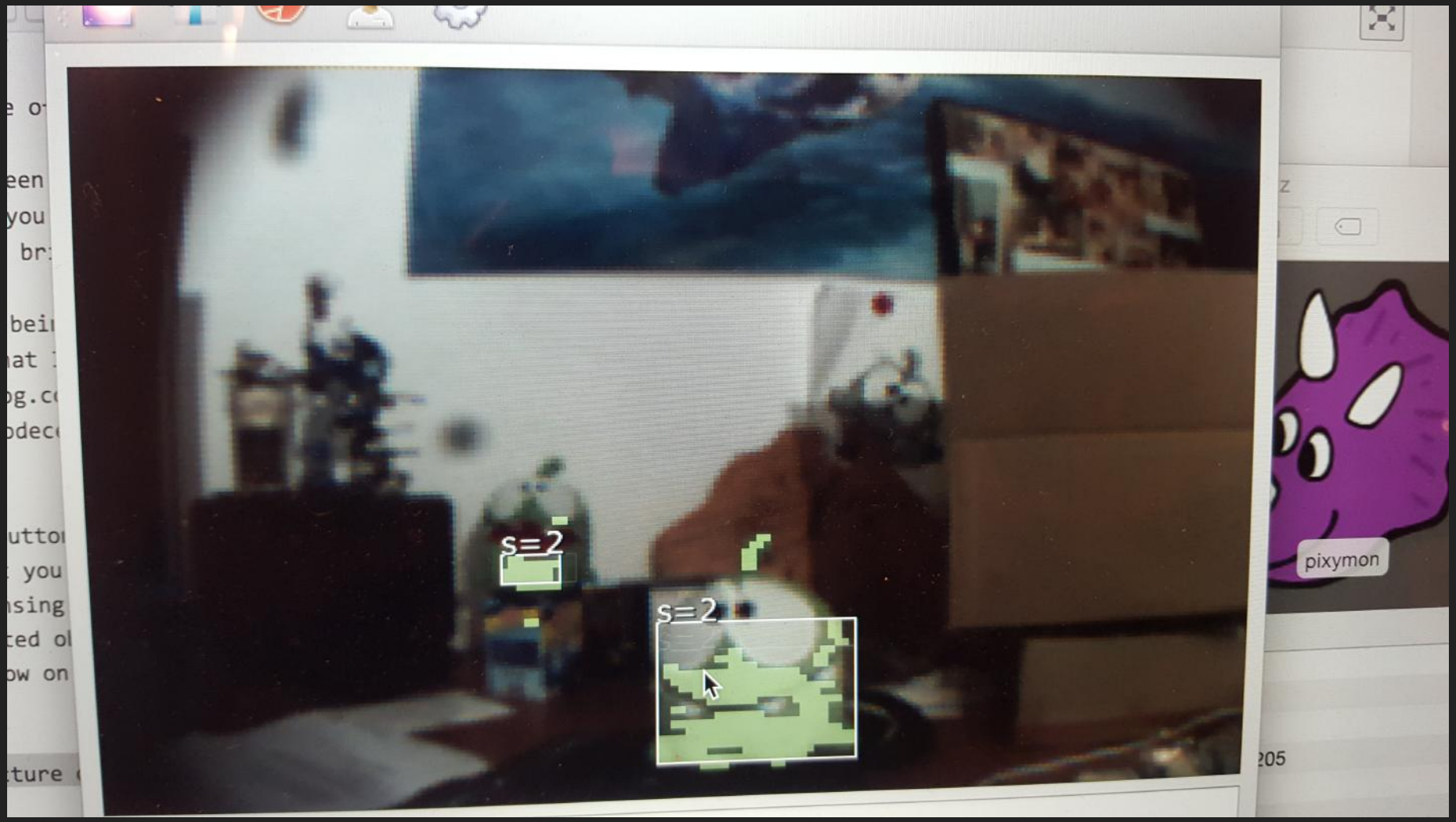


PRINTING THE CAM CARRIER

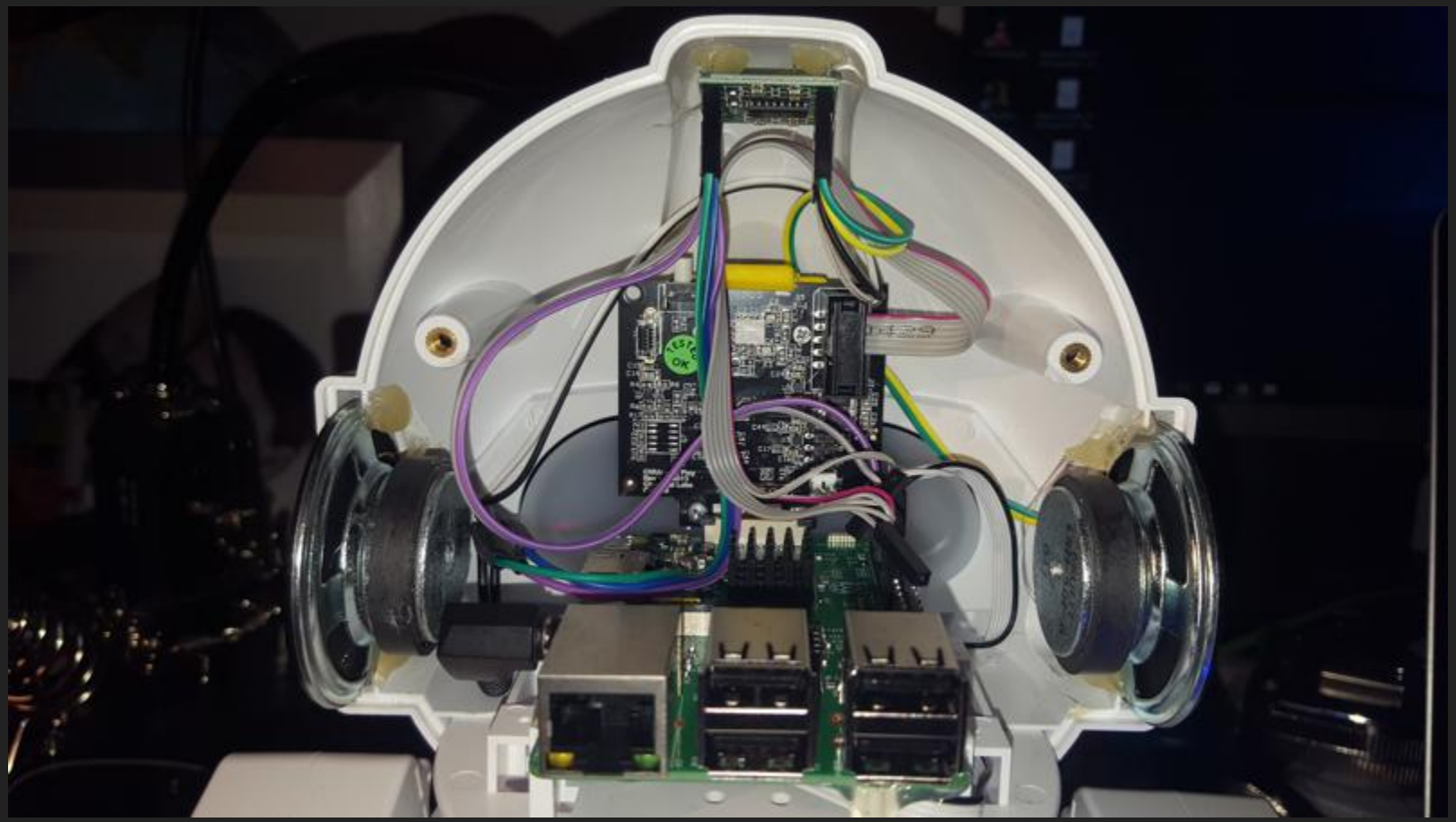


MOUNTED CAM

PIXY IN ACTION



FINISHED PARTS



THE END