

Webcamsteuerung mit Raspberry Pi

Philip Bell, B.Sc. Mathematik, 7. Semester - **Johannes Visintini**, B.Sc. Angewandte Informatik, 7. Semester

Betreut durch: Markus Kurz bzw. Gero Plettenberg und Thomas Kloepfer

Vorwort:

Dies ist eine Kurz-Präsentation unseres Praktikums. Für ausführlichere Informationen besuchen Sie bitte die Praktikumshomepage [1]. Uns erreichen Sie unter den folgenden E-Mail Adressen:

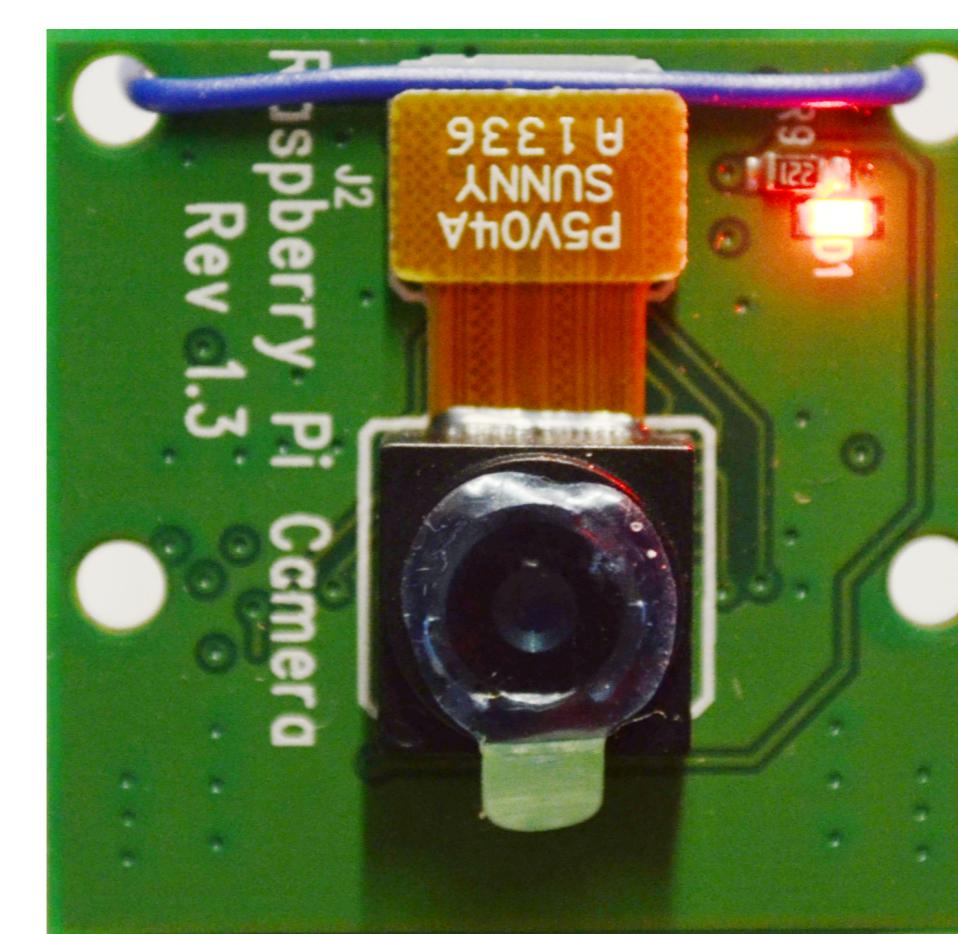
- philip.bell@web.de
- visintini@stud.uni-heidelberg.de

Aufgabenstellung:

Bau eines Gerüstes inkl. Schwenkvorrichtung für die Kamera Erstellung eines Webservers mit folgenden Funktionen:

- Anzeige eines Live-Streams
- UI zur Steuerung der Kamera

[1] <http://joanna.iwr.uni-heidelberg.de/rlab/de/home> → Projekte → SS2014 → Webcam Gruppe B



Haltevorrichtung:

Der Zweck dieses Teils ist einerseits alle anderen Segmente der Konstruktion zusammenzuhalten; andererseits sollte er die Montage an einer Wand zu erlauben. Zudem sollte die Haltevorrichtung über einen stabilen Stand verfügen. Wir entschieden uns zu der für diese Zwecke funktionalen T-Form.

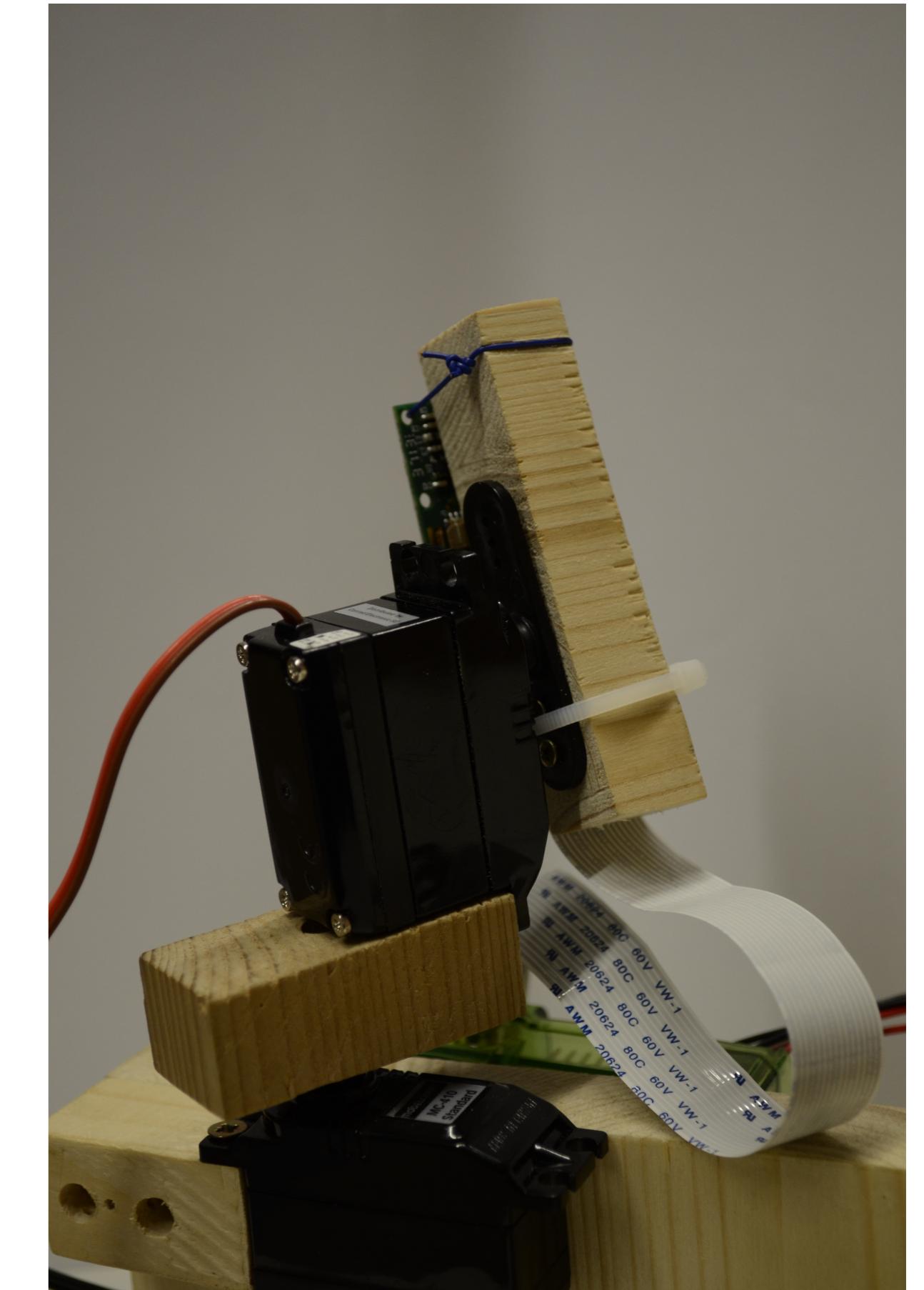
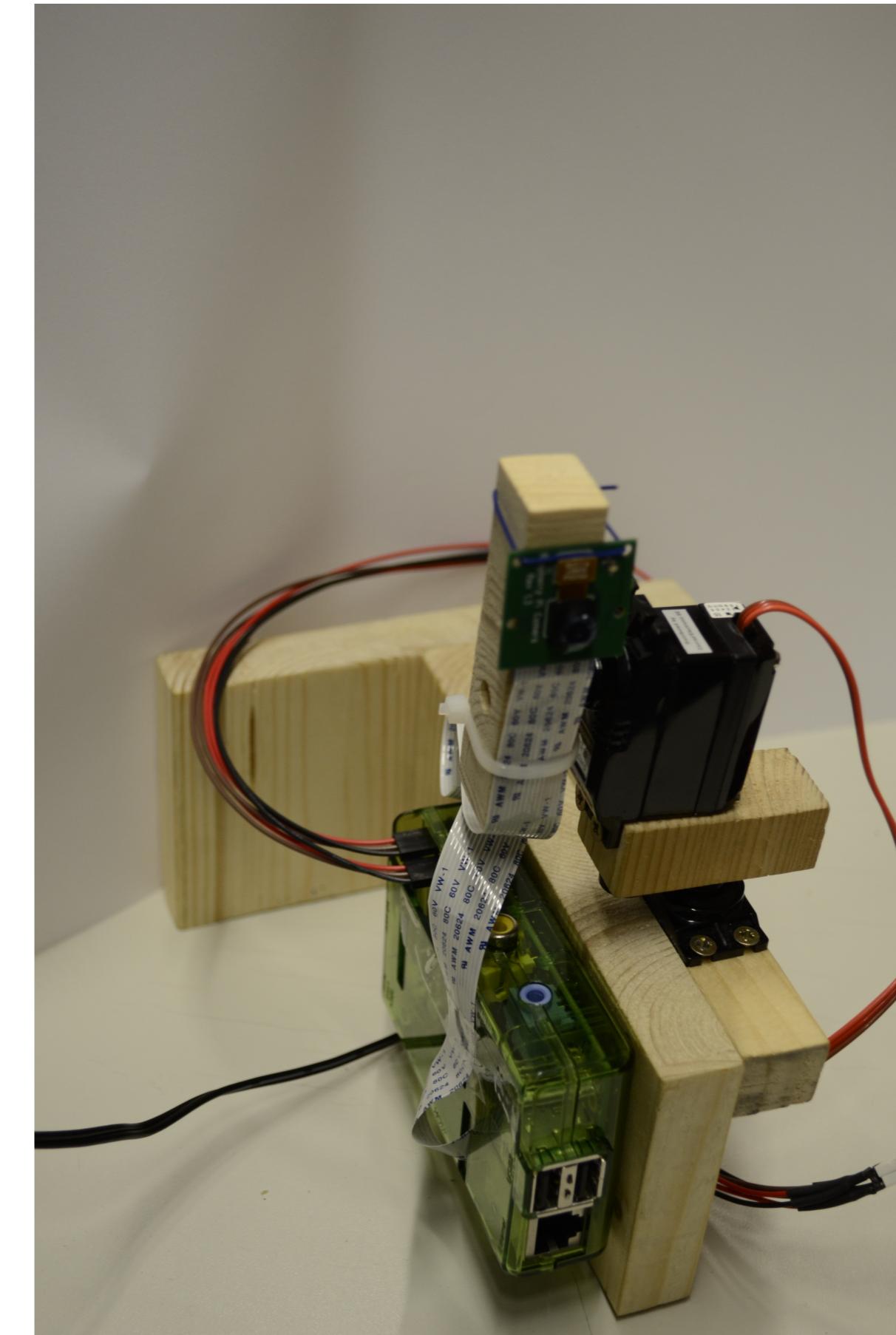
Schwenkvorrichtung:

Dieser Teil der Konstruktion ermöglicht die Ausrichtung der Kamera. Diese Funktionalität wird durch zwei Servomotoren unterstützt, von denen jeweils einer zur horizontalen und einer zur vertikalen Orientierung verwendet wird.

Hardware:

Wir haben die folgende Hardware benutzt:

- Raspberry Pi Model B Revision 2
- Raspberry Pi Camera Board
- Standard Analog Servo MC-410



Webserver:

Das Frontend ist mehrstufig aufgebaut. Der mJPG-Streamer mit Plugin für die rascam des Raspberry Pi (source) stellt den Stream in einer selbst gebauten HTML-Seite dar. Dieser wird als Dienst beim booten gestartet und läuft (unabhängig von den anderen Komponenten des Setups).

jQuery:

Die Webcam kann auf verschiedene Arten in horizontaler und vertikaler Richtung gedreht werden.

Es gibt drei Möglichkeiten:

Die *triviale*: Buttons

Die *intuitive*: Pfeiltasten

Die *mobile*: Wischen

Alle drei Möglichkeiten werden von Funktionen unter Zuhilfenahme der JavaScript-Bibliothek jQuery abgefangen. Im Hintergrund werden dann über diese Funktionen Anfragen an den Server gestellt, unter Übermittlung der gewünschten Bewegungsrichtung.

Die mobile Variante von jQuery sorgt gleichzeitig für ein schöneres (Grund-)Design der Webseite.

Apache:

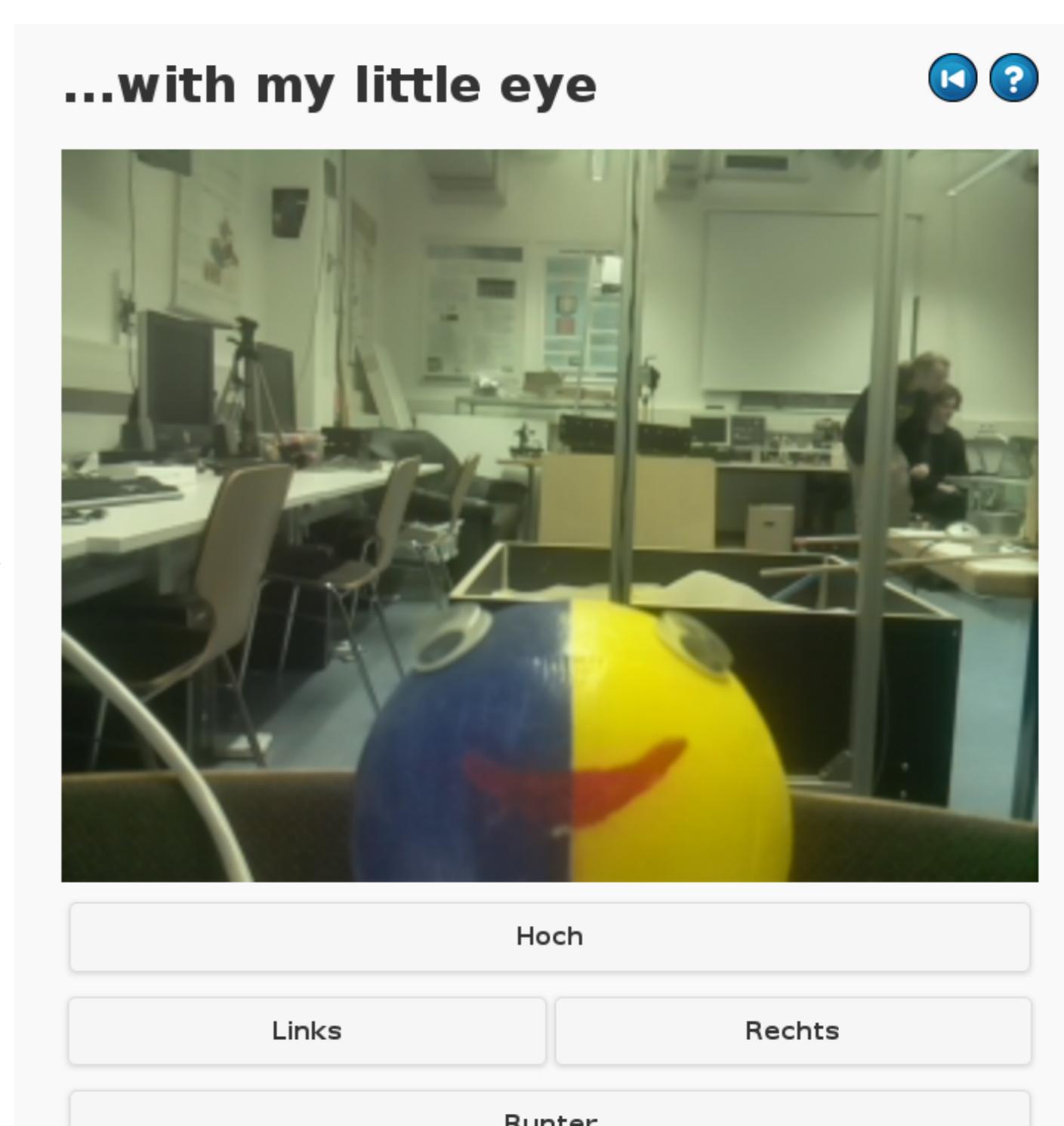
Als Webserver wird Apache verwendet, der als sehr stabil gilt. Dieser fungiert als Instanz, die mit dem Browser des Clients kommuniziert. Sämtlicher Datenverkehr (auch zum mJPG-Streamer) läuft über Apache. Der Webserver nimmt auch die Steuerbefehle entgegen und leitet sie über ein WSGI-Skript, das eine Verbindung zum moveserver aufbaut, an denselben weiter. Dieses WSGI-Skript überprüft auch, ob die Bewegungsrichtung erlaubt ist. So können falsche Werte nicht an den moveserver weitergegeben werden.

Sicherheit:

Da der Raspberry Pi als ganz normaler Computer bzw. Server im öffentlichen Netz hängt, musste sich überlegt werden wie dieser gegenüber Angriffen aus dem Internet abgesichert wird.

Als Firewall wird die populäre Firewall *iptables* und zur einfacheren Konfiguration derselben *firehol* verwendet. Diese ist so konfiguriert, dass nur Anfragen aus dem Universitätsnetz, das als sicher gilt, akzeptiert werden. Außerdem sind nur die Dienste *http* (Port 80, für den Webserver) und *ssh* (Port 22, zur Konfiguration und Wartung) freigegeben.

Durch ständige Updates und die oben erwähnte Firewall wird gewährleistet, dass der Raspberry Pi im Internet mindestens so geschützt ist, wie die meisten anderen Server.



moveserver:

Der moveserver ist ein Python Skript, das beim Einschalten des Raspberry gestartet wird. Zuerst einmal stellt er sicher, dass die Servos zu Beginn in eine bekannte Stellung, die wir als die Standardposition bezeichnen, gebracht werden, da ein Auslesen der Position aus den Servos nicht möglich ist. Weiterhin füllt er die vom WSGI-Skript erhaltenen Requests in die Queue und startet den Worker (s.u.). Zusätzlich führt er die Suspend-Funktionalität als Subprozess aus und verwaltet deren Zeit-Variable.

Queue & Worker:

Die eingehenden Requests werden in einer Queue (LIFO) der Länge 5 zwischengespeichert, dh die zuerst eingehenden Requests werden zuerst vom Worker abgearbeitet. Ist die Queue voll, wird der älteste Befehl verworfen. Diese Form der Priorisierung hat sich als funktional erwiesen: Auch in einer Testsituation, in der ein Testskript massiv viele Requests sendete, konnte ein Mensch die Steuerung, wenngleich nur mit intensivem Einsatz, beeinflussen. Hier folgt die vom Worker aufgerufene Funktion move():

```
def move(curr4, curr17, direction, step=3):  
    # Abfangen ungültiger Eingaben  
    if not direction in ("left", "right", "up", "down", "defaultpos"):  
        print "You should use left, right, up or down.\n";  
    return;  
    # tmp: position to move  
    # nr: Number of the GPIO-pin. 4 = vertical move, 17 = horizontal move.  
    # step: Schrittweite
```

```
tmp = None;  
nr = None;  
step = 10*step;  
if direction == "left":  
    print "move %s" % direction;  
    # Festlegen welcher Servo bewegt wird  
    nr = 4;  
    # Überprüfen ob der maximale Ausrichtungswinkel nicht überschritten wird  
    # Die Zahl gibt dabei die Signallänge, mit der die Servos angesprochen  
    # werden, in Mikrosekunden an  
    # Wir lassen eine Signallänge von 1000 bis 2000 zu; die lässt etwa eine  
    # Drehung um 90 Grad in beide Richtungen von der Mittelstellung (1500) zu  
    if curr4 + step <= 2000:  
        curr4 += step;  
    else:  
        curr4 = 2000;  
    tmp = curr4;  
    # [...] hier folgen dann die entsprechenden Abfragen für Rechts, Oben und Unten sowie Defaultpos  
    # Fehlermeldung bei falscher Eingabe  
else:  
    print "false usage of move();"  
    # Drehen des entsprechenden Servo (nr) zum angegebenen Ausrichtungswinkel (tmp)  
    go_servo(nr, tmp);  
    # Rückgabe der neuen Positionen der Servos  
return curr4, curr17
```