Stupid Cities - motion visualisation and kerb detection writeup

The Stupid Cities project sought two uses of motion sensing and capture as part of the aim to document difficulties faced by citizens with assisted mobility.

Brief: using appropriate micro controller, sensors and accessories, develop a solution to

- visualise motion (as detected by sensors) for use in early workshops to demonstrate the capabilities with participants.
- detect and collect data about dropped kerbs and other obstacles that affect mobility and access for users of wheelchairs (and other mobility aids).

Initial investigations.
After a look at available micro controllers, with combinations of sensors and accessories that might suit sensing, collection and connectivity with computers, the most promising device was the M5 stick. This contains an ESP32 Arduino controller with battery, small LCD screen, and a built in 6 axis IMU (inertial measurement unit - a combined accelerometer and gyroscope). It also has wifi and bluetooth capabilities built into the ESP32. This was appealing as it is full-featured, incredibly low cost and self contained. It does not have onboard GPS but a separate unit is available that can be connected via the available grove connector (wired for UART interface mode).

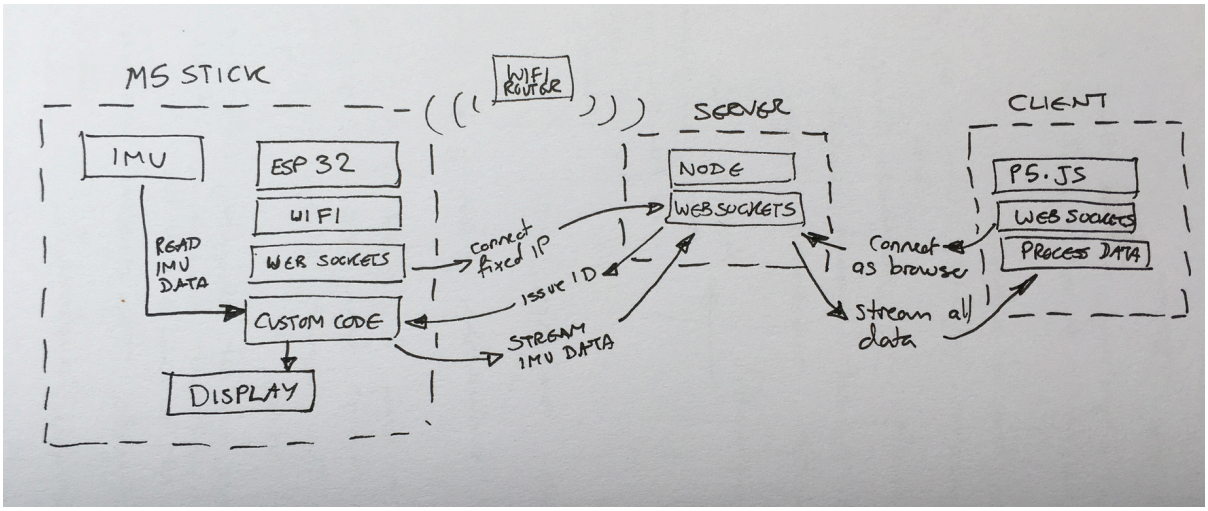
Technology stack for development
The M5stick can easily be programmed using the standard Arduino IDE. As memory is limited, only the most basic interface reading, processing and data handling can be carried out on board. Accessing the onboard IMU motion data is trivial, but to be useful the data must be passed to some other device. Three options exist - wired (via USB-C), Bluetooth and wifi. Serial over USB is great for testing and troubleshooting but wireless options give us much more flexibility.
Bluetooth was tested, and data was successfully received to a test device. Wifi was tested, sending data via a web socket connection to a device.
Adding the wifi stack and then a web socket client to the M5Stick is extra overhead, but gives us a pretty sophisticated connection for data sharing in real time, and seemed worthwhile for both early and future experiments.

In order to achieve a wifi connection, the wifi network SSID, security passcode, and the IP address of the target web socket server all need to be hard-coded into the program loaded onto the Arduino. This limited flexibility, but for the lab/testing this is not a problem. A node server, running on a fixed IP address, and running web sockets, provides the target connection for the device. It is programmed to listen for web socket requests, and to issue a “client” ID number to a successfully connected device.

The Arduino code waits for a wifi network to be detected upon boot, then it authenticates. The build in display allows us to inform the user that device has found a network, and that it has an assigned IP address. It then requests a web sockets connection, and upon

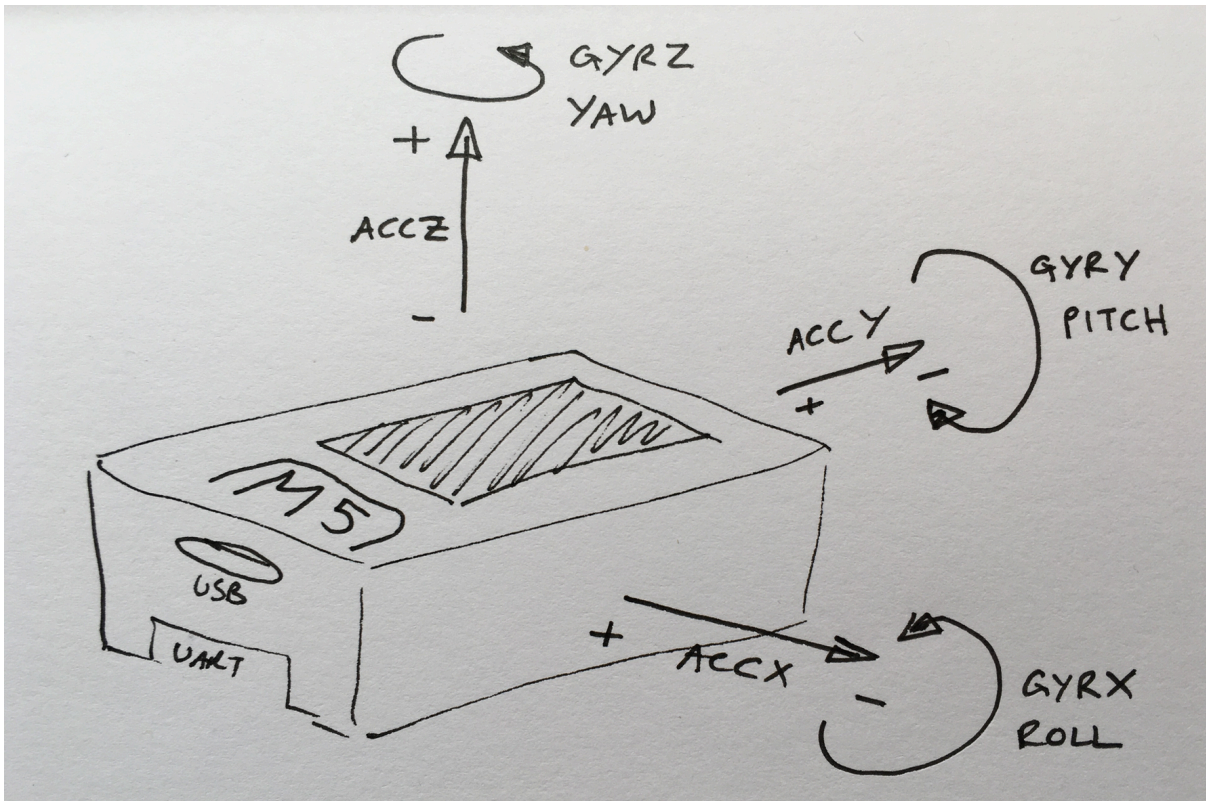
success is issued a client ID which it then displays.
From this point, the program streams current IMU motion data to the serve over web sockets.

This basic architecture allows multiple connected M5Sticks to connect to the server and stream their realtime motion data. The server collects this data, which can be used for a variety of purposes. Web clients can connect to the web socket server to retrieve the stream of IMU data from multiple devices and process it as needed.

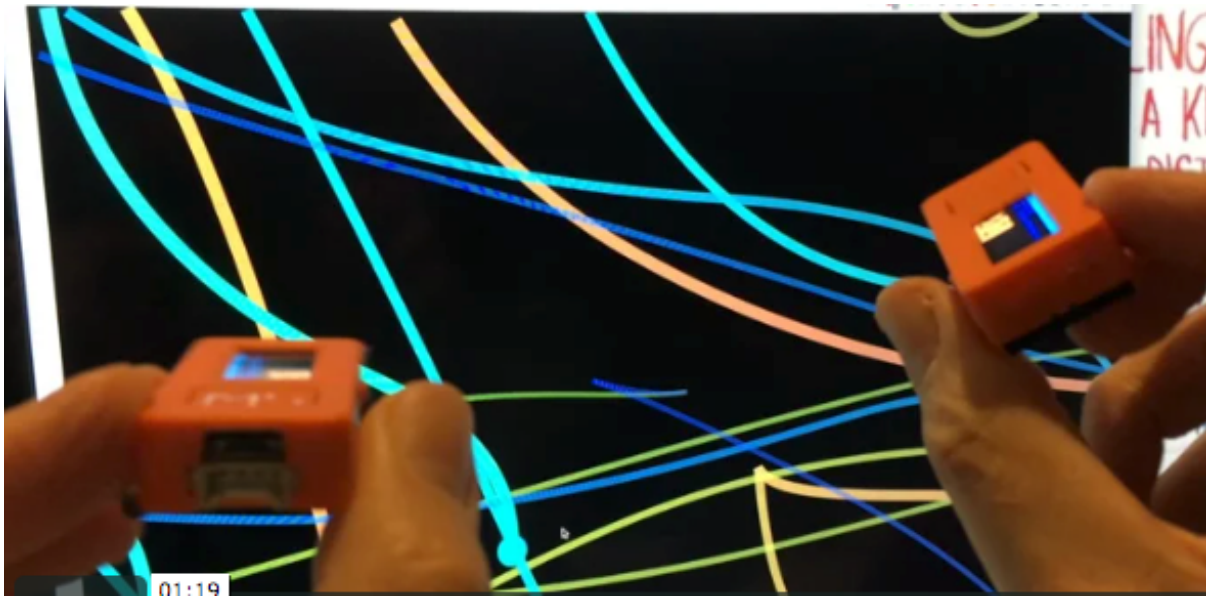


Motion visualisation

For the visualisation part of the brief, the arrangement above is implemented. The web client, running p5.js, takes in streams of IMU data. Each data packet contains an identified data value (for example Accelerometer in the Y-Plane is labelled ACCY, along with a single byte value, and the ID of the sending device).
As each new piece of IMU data is received, a table of data is built up for each device. if a new device is detected, a new data object is created for that device, which is updated with this and all subsequent IMU data for that device.



With a full set of IMU data available for each connected device, the visualisation software takes just the Y Accelerometer (forward backward acceleration) and Z rotation (left and right tuning forces) and integrates these using some basic physics simulation to give an approximate proxy to move a particle on the screen for each detected device.
Multiple visualisation styles are experimented with.

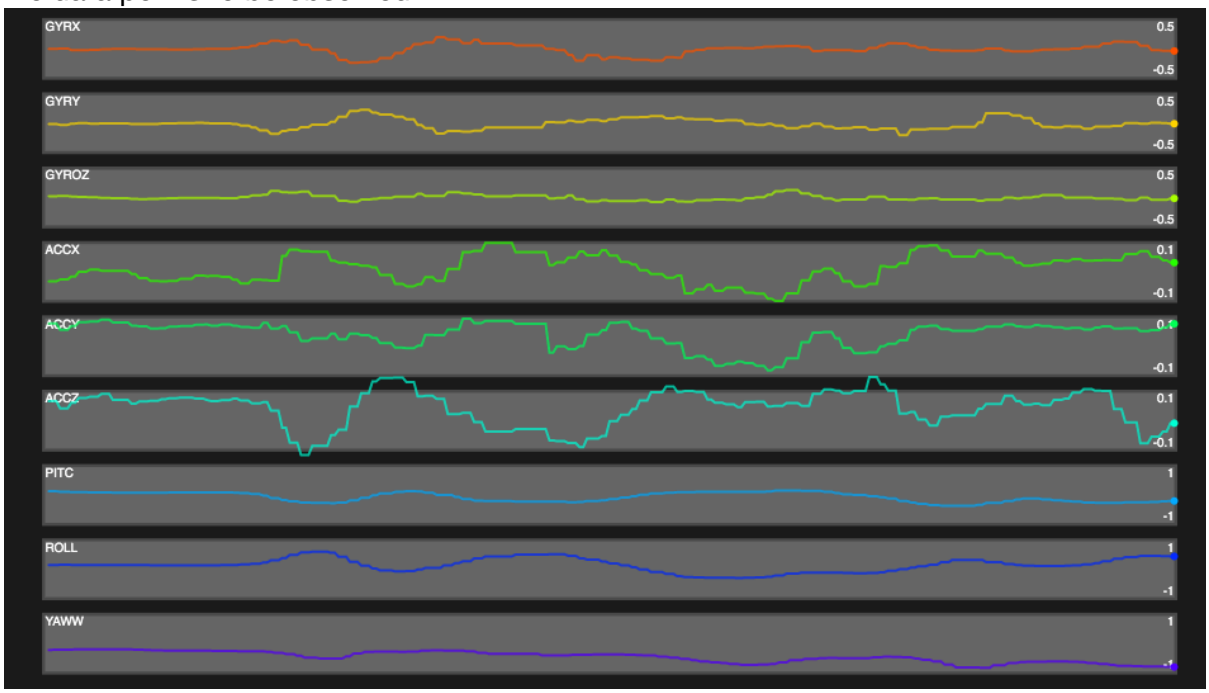


https://vimeo.com/393686077

Slope detection

To move on to kerb detection, a greater range of motion needed to be examined.
There are various proprietary algorithms that integrate full IMU data to provide noise-free, accurate estimates of angle, speed, even relative position. These are based upon mathematical principles. Articles were found for some of these but they were too specialised and detailed to enable rapid prototyping.
A much simpler integration of available data points was attempted based on basic geometry and the physics of simple motion.

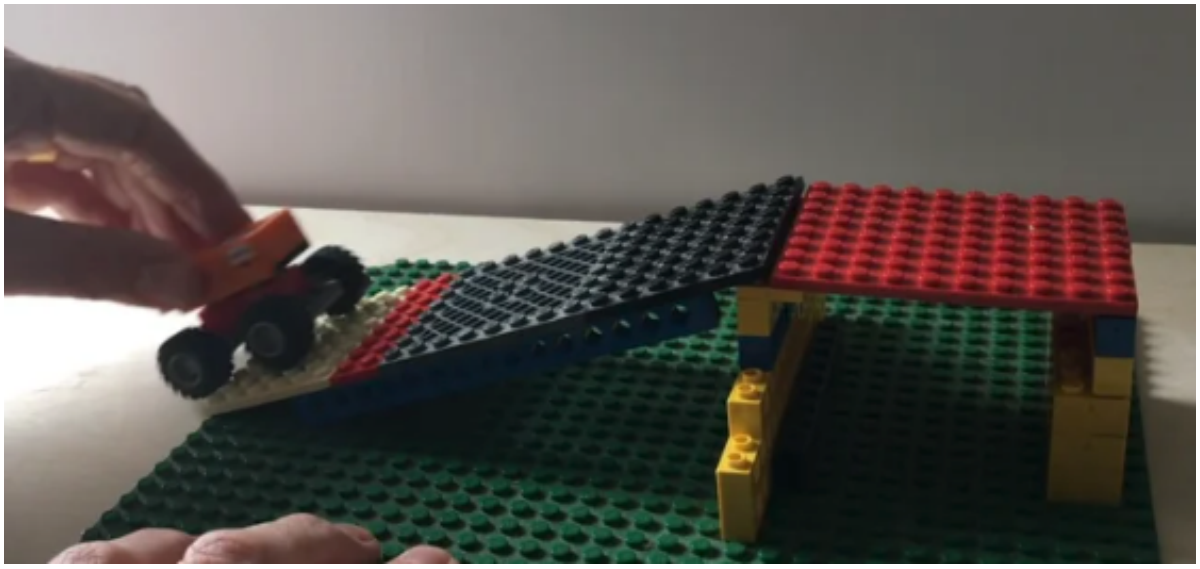
A p5 sketch was created to visualise the IMU data from one connected client. This allowed the relationship between any motion and the instantaneous effect on any one of the data points to be observed.



To provide a usable workflow for capturing and reading slope data, there are two web apps, one used for receiving captured slope data from the M5Stick, and writing it to database. A separate web page then retrieves this data and processes it for slope estimation.
A feature was added to the M5Stick code to start/stop data capture, and to add a time stamp. This is not an absolute timestamp, as the M5stick has no awareness of actual time, but elapsed time since the code started running gives us a relative time for each reading that can be used in calculate rates of change.
A corresponding p5 sketch used this capture start/stop feature to isolate a relevant subset of IMU data. When capture is “stopped” by the M5Stick, and that signal is received by the sketch all that timed datapoint are written to a record in a firebase database. This gives us a set of timed IMU data points captured by the device.
A second P5 sketch retrieves (for now the most recent) set of captured data from firebase, and processes this to render a representation of the slope, and estimate the angle.

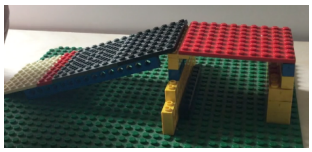
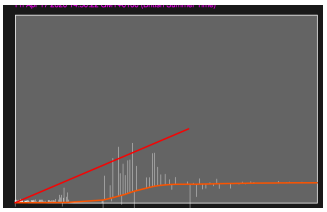
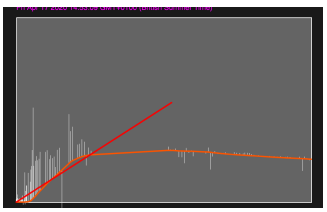
During lockdown, running slope detection experiments of this arrangement with a wheelchair were not possible.
A set of experiments were conducted with lego model and a variety of model slopes.

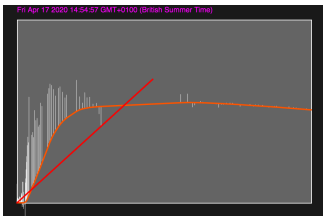
There are a few flaws with this approach. Firstly the scale between the model and detection accuracy of the M5stick would amplify errors. In addition, the relative size fo a kerb or slope compared to a chair wheel diameter would have a significant influence on the slope detected from the chair. Therefore these experiments test a principle and may not translate the real world.



https://vimeo.com/408876854

Tests were carried out using the capture arrangement (M5stick, local wifi network, and a node server running on a laptop connected to the same wifi network and the internet). This was used to capture the IMU data running the model up a variety of slopes. This data was written to a firebase database.
Separately the firebase data was retrieved using the second P5 sketch to visualise and estimate the slopes.

Actual slope	Slope rendered from processed data	Measured Angle	Calculated from data
		18.6°	23.1°
		25.9°	32.6°
		42.5°	42.5°

While the resulting graphs do reflect the difference in slopes, the angle estimates were not accurate or reliable. It would be possible to grade slopes (say a 1-5 scale of steepness). Further experiments would be needed on a variety of slopes and possibly with mobility aids with different wheel size and different wheel base (as these will both have a significant effect on the slope detection from an onboard device).

Next Steps: Moving from the lab to the real world.
The detection setup (with the M5 stick, local wifi, local server, local web app, firebase storage) is a cumbersome arrangement and would not suit real-world use. The ideal would be to isolate the M5stick and have the capture (as triggered by the start/stop capture button) write data to an SD card, or send data to a bluetooth connected mobile app). Neither of these options were investigated. The M5Stick is not able to write to an SD card without some modification. While SD card accessories are available, the pins on the ESP32 needed for writing to an SD card in standard Arduino use are used by the display, and some physical and software modification would be needed.
The M5Stack (the bigger and original device on which the M5 stick is based) has a built in SD card slot, a larger screen and more buttons, which could make for a more friendly interface.
Locating captured slope data with GPS, or with hand-recorded location information would need some consideration as well.
Stored SD card capture data, combined with location information could be uploaded to a firebase (or other) database that the Stupid cities web page and/or app could integrate with other information.

Lab setup
For the lab setup a TP-Link AC750 wireless access router is setup as a wifi router with SSID **crispy** and passcode **peppersalt**.
This router also acts as DHCP server.
A reserved address is setup for the MacBook that acts as server, 192.168.0.2
This address, along with the SSID and passcode are coded into the M5Stick code. They will need to be changed in a different lab setup.

Code locations
Repo: RealtimeM5data within the Stupid-Cities Git Project

Server
Node server to serve web socket connections and stream data to web clients.
SCWebSockets/webSocketServer

M5Stick
M5Stick code to connect to web socket server and stream timed IMU data to it.
SC_M5Stick/M5StickIMUWebsocketsSurfaceCapture/M5StickIMUWebsocketsSurfaceCa

pture.ino

Web clients
Various web client pages visualise the data in different ways.
SC_P5JSclient/GraphData - simple graphs of incoming IMU data
SC_P5JSclient/Visualisations/ - various visualisations based on incoming IMU data from multiple devices.
SC_P5JSclient/SurfaceCaptureTest - graphing, plus writes captured dataset to firebase when a capture-off is received
SC_P5JSclient/SurfaceDataViz2 - loading of most recent firebase data with slope visualisation.

To run the server, have a terminal session on a laptop connected to crispy wifi network with IP 192.168.0.2
change to the directory SCWebSockets/webSocketServer and run

```
node index.js
```

Load the above .ino file in the arduino IDE and push it to the M5stick.
With this node server running on the crispy wifi network, a client M5stick running the right code will connect and be assigned a web socket client number, which is displayed.
You should immediately see incoming IMU data from this client in the terminal window.
<images - m5 disc, conn, and termminal>
You can then run any of the above p5 sketches above by running its index.html file. none of the features should cause cross-site problems by running serverless. If they do, you can run them by running the python http server in the holding directory (in a terminal window, change to the holding directory and run

```
python3.-m http.server
```

Dave Webb
June 2020
dave.webb@cranbim.com
@crispysmokedweb