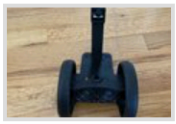
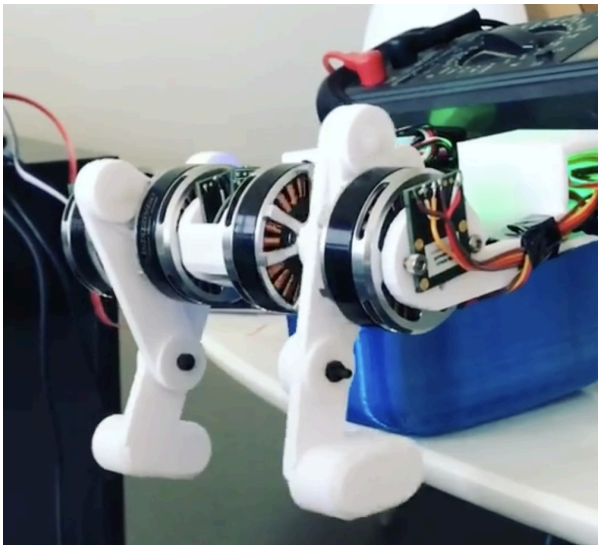
Teleport: Telepresence and music player robot.

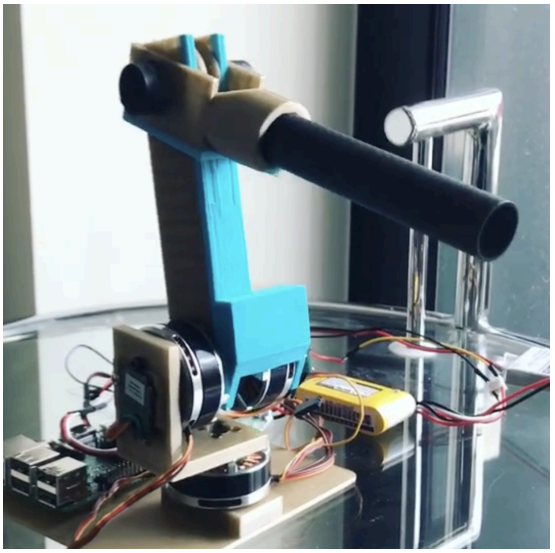
Teleport is a music player with wheels, and a two-way audio telepresence link with camera, controlled from the web.

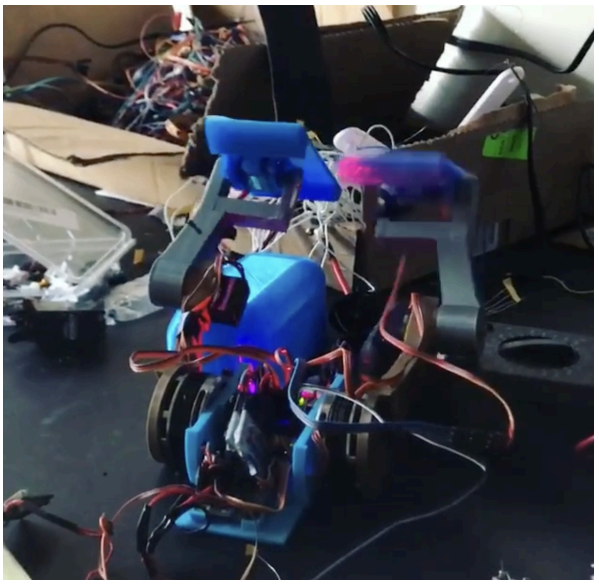
Designed by Teleport in United States of America



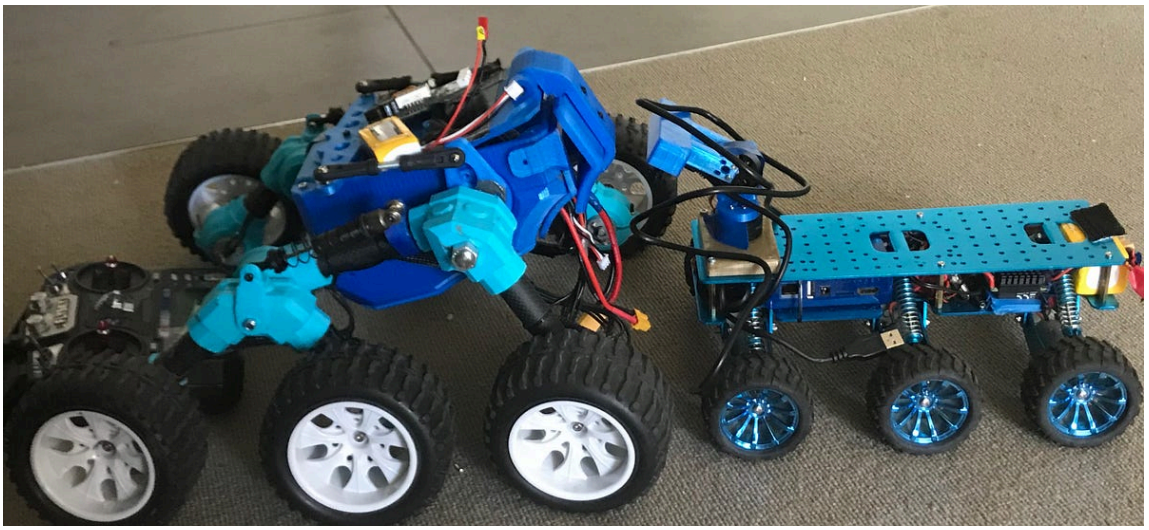
Teleport is a music player with wheels, and a two-way audio telepresence link with camera, controlled from the web.

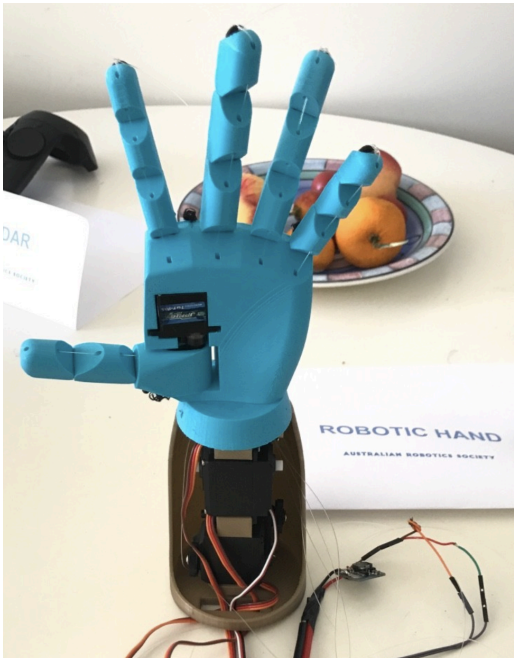


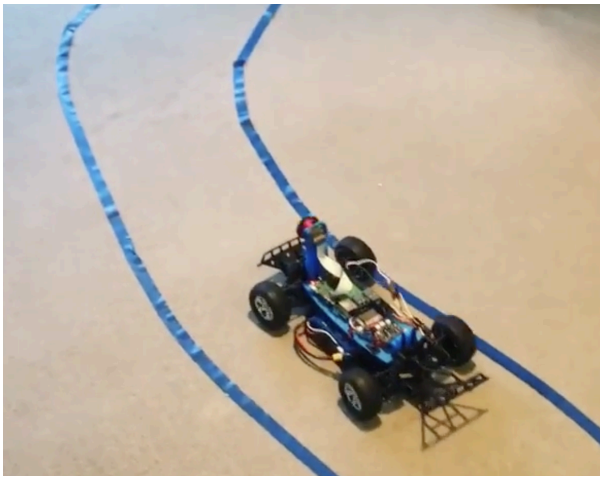





GBM4008
GBM4008
MORN SU
CUTTING
45 X 30







www.instagram.com/australianroboticssociety

Look, it's a robot!

Tom Jacobs
20 subscribers

Analytics | Edit video

179 views 12 years ago
The Rover goes on its first trip outside. ...more

Tom Jacobs

- Currently building robots at Ghost Robotics:
    - Built iterative prototype robots over six years.
    - Developed the robot controller Android app.
    - Designed and built the joystick remote control.
    - Maintained the SDK developer docs and tools.
    - Maintained network comms protocols for robots.
    - Developed robot health diagnostics software, app and Linux.
    - Upgraded hardware and software of robots at customer sites.
    - Handled the customer support.
    - Trained customers on robots.
    - Wrote the robot user guides.
    - Profiled motor performance.
    - Wrote robot firmware.
- C, C++, Bash, Python, JavaScript, Android Java.
- Completed Coursera's *Machine Learning* course.
- Completed Udacity's *Self Driving Car Engineer* nanodegree.
- Created the Expensify mobile apps, growing company from four people.
- Rebuilt and ran a network of 1m P2P clients at Red Swoosh
- Built a remotely piloted internet connected robot.
- Founded an ecommerce consumer electronics business.
- Co-founded an analytics business.

```
// --- GStreamer ---
void* gstreamerThreadFunction(void* arg)
{
    while (true)
    {
        // Start GStreamer streaming video?
        if (gstreamer) {

            // Standard
            int width = 640;
            int height = 480;
            int framerate = 25;

            // Construct pipeline
            if (LOCAL) gstreamer_sink = "127.0.0.1";
            int port = (10000+DEVICE_ID);
            string pipeline = "gst-launch-1.0 rpicamsrc preview=false bitrate=" + to_string(bitrate) + " keyframe-interval=30 ! \
video/x-h264, framerate=" + to_string(framerate) + "/1, profile=baseline, width=" + to_string(width) + ", height=" + to_string(height) + " ! \
h264parse ! \
rtph264pay config-interval=1 pt=96 mtu=500000 ! \
udpsink host=" + gstreamer_sink + " port=" + to_string(port);
            printf("Starting GStreamer video.\n");
            printf("%s\n", pipeline.c_str());
            system(pipeline.c_str());
        }

        // Wait
        usleep(1E5);
    }
    return 0;
}


// Thread to read voltage
void* voltageThreadFunction(void* arg)
{
    // Read via ADC
    while (true)
    {
        // Single channel read command
        int adc_channel = 0;
        char command = 0b11000000;         // Start bit, single channel read
        command |= (adc_channel & 1) << 5; // Channel number

        // The bottom three bits of command are zero; this is to account for the
        // extra clock to do the conversion, and the low null bit returned at
        // the start of the response
        unsigned char bufferSPI[3];
        bufferSPI[0] = command;
        bufferSPI[1] = 0;
        bufferSPI[2] = 0;
        int sent = wiringPiSPIDataRW(0, bufferSPI, 3);

        // Parse out the 10 bits of response data and return it
        int result = (bufferSPI[0] & 0x01) << 9;
        result |= (bufferSPI[1] & 0xFF) << 1;
        result |= (bufferSPI[2] & 0x80) >> 7;
        result &= 0x3FF;

        // Convert to voltage
        float voltage = (result + BIAS)/SCALE_FACTOR;
        if (result == 0) voltage = 0.0;

        // Filter
        if (voltage > 0 && voltage < 6)
        {
            if (!batteryVoltage) batteryVoltage = voltage;
            else batteryVoltage = (batteryVoltage * BATTERY_FILTER) + voltage * (1.0f - BATTERY_FILTER);
        }


// --- Buttons ---

void* interfaceThreadFunction(void* arg)
{
    if (!GPIO_enabled) return 0;

    while (true)
    {
        // Read buttons
        button[0] = !digitalRead(4);
        button[1] = !digitalRead(12);
        button[2] = !digitalRead(16);
        button[3] = false; //!digitalRead(17); // Faulty, fixed next board revision

        // Fire event if changed
        for (int i = 0; i < NUM_BUTTONS; i++)
            if (button[i] != buttonPrevious[i])
                buttonEvent(i, button[i]);

        // Save state
        for (int i = 0; i < NUM_BUTTONS; i++)
                buttonPrevious[i] = button[i];

        // Wait 100ms, checking buttons 10 times a second
        usleep(1E5);
    }
}
```