## Project File Tree

```
.
├── .gitignore
├── Session.vim
├── build.sh
├── include
│   ├── README
│   └── extrakeys.h
├── lib
│   └── README
├── pinout.png
├── platformio.ini
├── set_hwids.py
├── spacemousev1.jpg
├── spacemousev2.png
├── src
│   └── main.cpp
└── test
    └── README
```

## platformio.ini

```
; PlatformIO Project Configuration File
;
;   Build options: build flags, source filter
;   Upload options: custom upload port, speed and extra flags
;   Library options: dependencies, extra library storages
;   Advanced options: extra scripting
;
; Please visit documentation for the other options and examples
; https://docs.platformio.org/page/projectconf.html
[env:micro]
platform = atmelavr
board = micro
framework = arduino
extra_scripts = pre:set_hwids.py
monitor_filters = send_on_enter
monitor_eol = CRLF
monitor_echo = yes

[platformio]
src_dir = src
```

## set_hwids.py

```
# This script sets the correct names and hardware IDs for Spacemouse emulation
 ↪  to work.
```

```
# From https://stackoverflow.com/a/76049354.

Import("env")
board_config = env.BoardConfig()
board_config.update("build.hwids", [["0x256f", "0xc631"]])
board_config.update("build.usb_product", "SpaceMouse Pro Wireless (cabled)")
board_config.update("vendor", "3Dconnexion")
```

### src/main.cpp

```
#include <HID.h>
#include <string.h>

/*
 * HID descriptor that matches Space Navigator Pro and send_command() functions taken from here:
 */

#define ANALOG_MAX 4096 // measuring range of ADCs
#define THRESHOLD 512 // "deadzone" size. any input below this will be ignored
#define FILTERING 1  // number of rolling average slots. higher values reduce noise at the cost of input lag
#define DELAY 8 // use this to control the main loop speed
#define SPEED 1/16

static const uint8_t _hidReportDescriptor[] PROGMEM = {
    0x05, 0x01,           //  Usage Page (Generic Desktop)
    0x09, 0x08,           //  0x08: Usage (Multi-Axis)
    0xa1, 0x01,           //  Collection (Application)
    0xa1, 0x00,           // Collection (Physical)
    0x85, 0x01,           //  Report ID
    0x16, 0x00, 0x80,     //logical minimum (-500)
    0x26, 0xff, 0x7f,     //logical maximum (500)
    0x36, 0x00, 0x80,     //Physical Minimum (-32768)
    0x46, 0xff, 0x7f,     //Physical Maximum (32767)
    0x09, 0x30,           //    Usage (X)
    0x09, 0x31,           //    Usage (Y)
    0x09, 0x32,           //    Usage (Z)
    0x75, 0x10,           //    Report Size (16)
    0x95, 0x03,           //    Report Count (3)
    0x81, 0x02,           //    Input (variable,absolute)
    0xC0,                 //  End Collection
    0xa1, 0x00,           // Collection (Physical)
    0x85, 0x02,           //  Report ID
    0x16, 0x00, 0x80,     //logical minimum (-500)
    0x26, 0xff, 0x7f,     //logical maximum (500)
    0x36, 0x00, 0x80,     //Physical Minimum (-32768)
    0x46, 0xff, 0x7f,     //Physical Maximum (32767)
    0x09, 0x33,           //    Usage (RX)
    0x09, 0x34,           //    Usage (RY)
    0x09, 0x35,           //    Usage (RZ)
    0x75, 0x10,               //    Report Size (16)
    0x95, 0x03,               //    Report Count (3)
    0x81, 0x02,               //    Input (variable,absolute)
    0xC0,                     //  End Collection

    0xa1, 0x00,               // Collection (Physical)
    0x85, 0x03,               //  Report ID
    0x15, 0x00,               //   Logical Minimum (0)
    0x25, 0x01,               //    Logical Maximum (1)
    0x75, 0x01,               //    Report Size (1)
    0x95, 32,                 //    Report Count (24)
    0x05, 0x09,               //    Usage Page (Button)
    0x19, 1,                  //    Usage Minimum (Button #1)
    0x29, 32,                 //    Usage Maximum (Button #24)
    0x81, 0x02,               //    Input (variable,absolute)
    0xC0,
    0xC0
};

// send function and HID descriptor as seen here:
 ↪  https://www.printables.com/model/864950-open-source-spacemouse-space-mushroom-remix
void send_command(int16_t rx, int16_t ry, int16_t rz, int16_t x, int16_t y,
 ↪  int16_t z) {
  uint8_t trans[6] = { x & 0xFF, x >> 8, y & 0xFF, y >> 8, z & 0xFF, z >> 8 };
  HID().SendReport(1, trans, 6);
  uint8_t rot[6] = { rx & 0xFF, rx >> 8, ry & 0xFF, ry >> 8, rz & 0xFF, rz >>
   ↪  8 };
  HID().SendReport(2, rot, 6);
}

bool button;
bool prev_button;
bool button_event;
bool enabled;

enum Pinout:byte
{ // current pin configuration
    STICK0_U = A1,
    STICK0_V = A0,
    STICK1_U = A2,
    STICK1_V = A3,
    STICK2_U = A7,
    STICK2_V = A6,
    STICK3_U = A8,
    STICK3_V = A9,
};


struct roll //raw channel input rotating buffer
{
    int index; //used for rotating buffer indexing
    int offset;//used for calibrating around 0
    int val[FILTERING];//actual buffer
};
```

```
void roll_update(struct roll *roller, int newval)
{
    roller->val[roller->index]=newval;
    roller->index=(roller->index + 1)%FILTERING;
}


int roll_avg( struct roll roller)
{
    int avg = 0;
    // pretty sure this can be done in a much more clever way, without using a
     ↪  for
    for (int i=0; i<FILTERING; ++i)
    {
        avg += roller.val[i];
    }
    avg /= FILTERING;
    return avg;
}

void roll_zero(struct roll *roller)
{
    roller->offset=roll_avg(*roller);
}

int normalize(struct roll roller)
{ // centers the potentiometer values around 0
    return roll_avg(roller)-roller.offset;
}

struct
{
    struct roll Au;
    struct roll Av;
    struct roll Bu;
    struct roll Bv;
    struct roll Cu;
    struct roll Cv;
    struct roll Du;
    struct roll Dv;
} raw_rolling;

void zero_all()
{
    roll_zero(&raw_rolling.Au);
    roll_zero(&raw_rolling.Av);
    roll_zero(&raw_rolling.Bu);
    roll_zero(&raw_rolling.Bv);
    roll_zero(&raw_rolling.Cu);
    roll_zero(&raw_rolling.Cv);
    roll_zero(&raw_rolling.Du);
    roll_zero(&raw_rolling.Dv);
```

```
}

struct uv
{ // we will work with 2d vectors a lot
    int16_t u;
    int16_t v;
};

struct
{ // normalized local coordinates of each sticks
    struct uv A;
    struct uv B;
    struct uv C;
    struct uv D;
}input;

struct
{ // movement increments on each motion
    int zoom;
    int roll;
    struct uv pan;
    struct uv orbit;
}motion;

struct active_channel
{ // whether the current channel has active input (more than threshold)
    bool zoom;
    bool roll;
    bool pan;
    bool orbit;
};

 // we need this for edge detection
struct active_channel curr;
struct active_channel prev;



void updateInput()
{ // read input values from analog ports

    roll_update(&raw_rolling.Au, analogRead(Pinout::STICK0_U));
    roll_update(&raw_rolling.Av, analogRead(Pinout::STICK0_V));
    roll_update(&raw_rolling.Bu, analogRead(Pinout::STICK1_U));
    roll_update(&raw_rolling.Bv, analogRead(Pinout::STICK1_V));
    roll_update(&raw_rolling.Cu, analogRead(Pinout::STICK2_U));
    roll_update(&raw_rolling.Cv, analogRead(Pinout::STICK2_V));
    roll_update(&raw_rolling.Du, analogRead(Pinout::STICK3_U));
    roll_update(&raw_rolling.Dv, analogRead(Pinout::STICK3_V));

    input.A.u = normalize(raw_rolling.Au);
    input.A.v = normalize(raw_rolling.Av);
    input.B.u = normalize(raw_rolling.Bu);
    input.B.v = normalize(raw_rolling.Bv);
    input.C.u = normalize(raw_rolling.Cu);
    input.C.v = normalize(raw_rolling.Cv);
    input.D.u = normalize(raw_rolling.Cu);
    input.D.v = normalize(raw_rolling.Cv);
    input.D.u = normalize(raw_rolling.Du);
    input.D.v = normalize(raw_rolling.Dv);
}

void plotInputs()
{
    Serial.print(input.A.u);
    Serial.print("\t");
    Serial.print(input.A.v);
    Serial.print("\t");
    Serial.print(input.B.u);
    Serial.print("\t");
    Serial.print(input.B.v);
    Serial.print("\t");
    Serial.print(input.C.u);
    Serial.print("\t");
    Serial.print(input.C.v);
    Serial.print("\t");
    Serial.print(input.D.u);
    Serial.print("\t");
    Serial.print(input.D.v);
    Serial.print("\r\n");
}

void printInputs()
{
    Serial.print("INPUT:\t ");
    Serial.print("\tAu=\t ");
    Serial.print(input.A.u);
    Serial.print("\tAv=\t ");
    Serial.print(input.A.v);
    Serial.print("\tBu=\t ");
    Serial.print(input.B.u);
    Serial.print("\tBv=\t ");
    Serial.print(input.B.v);
    Serial.print("\tCu=\t ");
    Serial.print(input.C.u);
    Serial.print("\tCv=\t ");
    Serial.print(input.C.v);
    Serial.print("\tDu=\t ");
    Serial.print(input.D.u);
    Serial.print("\tDv=\t ");
    Serial.print(input.D.v);
    Serial.print("\r\n");
}

void printMotions()
{
    Serial.print("INPUT:   ");
    Serial.print("\tZoom=\t ");
    Serial.print(motion.zoom);
    Serial.print("\tPanX=\t ");
    Serial.print(motion.pan.u);
    Serial.print("\tPanY=\t ");
    Serial.print(motion.pan.v);
    Serial.print("\tOrbitX=\t ");
    Serial.print(motion.orbit.u);
    Serial.print("\tOrbitY=\t ");
    Serial.print(motion.orbit.v);
    Serial.print("\r\n");
}

void plotMotions()
{
    Serial.print(motion.zoom);
    Serial.print("\t");
    Serial.print(motion.pan.u);
    Serial.print("\t");
    Serial.print(motion.pan.v);
    Serial.print("\t");
    Serial.print(motion.orbit.u);
    Serial.print("\t");
    Serial.print(motion.orbit.v);
    Serial.print("\t");
    Serial.print(motion.roll);
    Serial.print("\r\n");
}

/*
 * THIS IS WHERE THE MAGIC HAPPENS
 * everythig else is mostly just boilerplate for reading values
 * and simulating keyboard&mouse inputs
 * This is where the kinematic expressions calculate
 * the movement on all channels of the 3D transform
 * from each stick's local UV space
 */

void calcMotion()
{ // calculates the motions from the states of the joysticks
    motion.zoom = (input.A.v + input.B.v + input.C.v + input.D.v)/4;
    motion.roll = (input.A.u + input.B.u + input.C.u + input.D.u)/4;
    motion.pan =
    {
        (input.C.u - input.A.u)/2,
        (input.B.u - input.D.u)/2
    };
    motion.orbit=
    {
        (input.C.v - input.A.v)/2,
        (input.B.v - input.D.v)/2
    };
    curr.zoom = (abs(motion.zoom) > THRESHOLD);
    curr.roll = (abs(motion.roll) > THRESHOLD);
    curr.pan = (abs(motion.pan.u)+abs(motion.pan.v) > 2*THRESHOLD);
    curr.orbit = (abs(motion.orbit.u)+abs(motion.orbit.v) > 2*THRESHOLD);
}
```

```
void apply_motion()
{
    send_command(motion.orbit.u, motion.orbit.v, motion.roll, motion.pan.u,
 ↪  motion.pan.v, motion.zoom);
}

void setup()
{

    static HIDSubDescriptor node(_hidReportDescriptor,
     ↪  sizeof(_hidReportDescriptor));
    HID().AppendDescriptor(&node);

    for (int i=0;i<FILTERING;i++)
    {
        updateInput();
    }
    zero_all();
    enabled = true;
//  Serial.begin(9600);
}

void loop()
{
    updateInput();
//  plotInputs();
    calcMotion();
//  plotMotions();
    if(enabled)
    {
        apply_motion();
    }
}
```