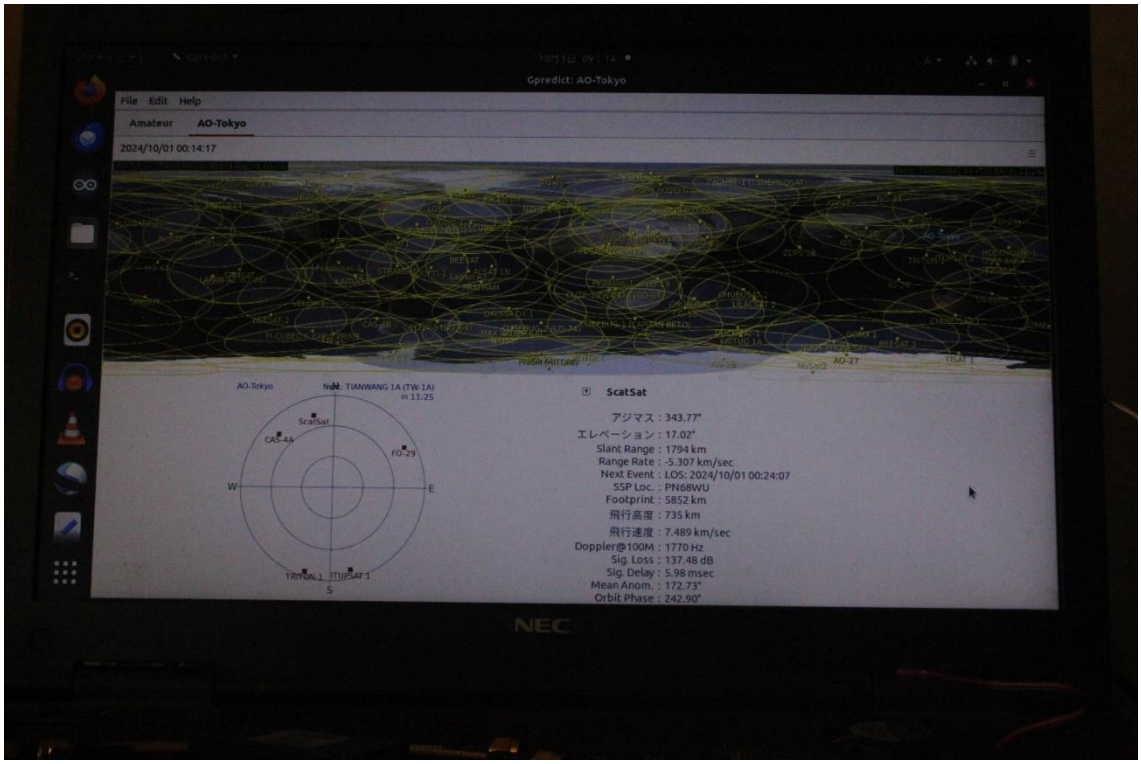
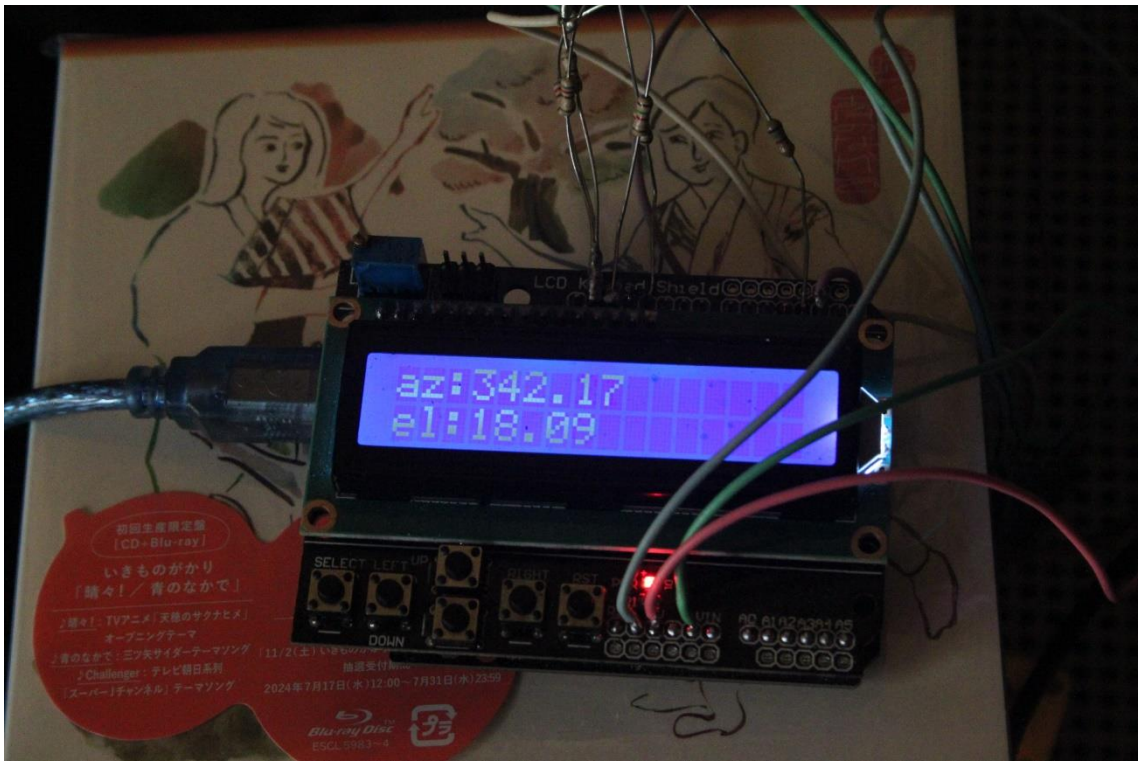
改造衛星追尾アプリケーション

On ubuntu オープンソース





1．はじめに

衛星追尾アプリケーションのオープンソースを発見しシリアル通信を埋め込んだ

2，概要

容易ではなかったがアルディーノダミー端末への表示完成

2．シリアル通信通信プロトコル RS232C

※アルディーノ UNO ダミースケッチ書き込み済接続前提

/dev/ttyACM0 のみ対応。いずれも文字列送信。通信レート 115200 ボー

プロトコル：

アジマス情報：

“az:アジマス情報＋改行” <- 文字列です

エレベータ情報：

“el:エレベータ情報＋改行” <- 文字列です

3．Ubuntu 側の設定

sudo adduser $USER dialout

シリアルポートを USB 接続するたびに権限付与させたい

Ubuntu で sudo chmod 666 /dev/ttyACM0 とコマンド入力することで権限が付与される。

ただし、再起動すると元の権限に戻ってしまうので接続するたびに自動で権限を付与させたい。

sudo vi /lib/udev/rules.d/50-udev-default.rules <- お気に入りのエディタを選ぶ

（　変　更　前　）KERNEL=="tty[A-Z]*[0-9]|pppox[0-9]*|ircomm[0-9]*|noz[0-9]*|rfcomm[0-9]*", GROUP="dialout"
（　変　更　後　）KERNEL=="tty[A-Z]*[0-9]|pppox[0-9]*|ircomm[0-9]*|noz[0-9]*|rfcomm[0-9]*", GROUP="dialout", MODE="0666"

４．インストール

※改造済衛星追尾インストール
※アルディーノ UNO + LCD シールドスケッチ書き込み済接続前提
１．https://github.com/csete/gpredict <- .zip ファイルをダウンロードする＆展開する
２．autogen.sh 実行
３．gtk-single-sat.c < src ディレクトリに上書き ※わたしの改造ソフトウエアモジュールです
４．./configure --prefix=/home/user/predict < ※user は自分の PC の名前
５．make
６．make install
※動くはずです

※ubuntu 上のアプリケーションです

５．液晶との通信プロトコル（RS232C）

・可変長
・LCD に直接書き込み
・エラーは発生しない
１．RS232C 通信プロトコルを設定する
（１１５２００ボーのノンパリのストップビット１）
２．送信側はアジマス、エレベータともに可変長の文字列データの無限送信
３．受信側の処理
３．１．受信側基本：無限受信

１．シリアル受信データを、ともかく液晶に表示する
２．文字 'a' が来たら画面クリア１行目１文字目にカーソル設定 <- 次から１行目１文字目から上書き表示する
３．文字 'e' が来たら２行目１文字目にカーソル設定 <- 次から２行目１文字目から上書き表示する

※バッファリングもしていません
※データの終わりの検出はない <- 文字のみを表示、ほかは無視 <- 表示しない
※通信エラー対策：USB 接続なので無視

アルディーノダミースケッチ：該当部分のみ：

```
ch = Serial.read();
switch(ch){
 case 0x0A:
     break;
 case 0x0D:
     break;
 case ':':
     lcd.write(ch);
     break;
 case 'a':
     setcuser0();
     lcd.clear();
     lcd.write(ch);
     break;
 case 'z':
     lcd.write(ch);
     break;
 case 'e':
     setcuser1();
     lcd.write(ch);
     break;
 case 'l':
     lcd.write(ch);
     break;
 case '0':
 case '1':
 case '2':
 case '3':
 case '4':
 case '5':
 case '6':
 case '7':
```

```
        case '8':
        case '9':
        case '.':
        case '-':
            lcd.write(ch);
            break;
    };
```

６．今後の課題

実際のアジマスローテーター制御とエレベータローテーター制御

７．追加定義及び追加モジュール及び改造モジュール

追加定義：

```
# define CRTSCTS    020000000000          /* flow control shinji-y added */
```

追加モジュール：

```
// shinji-y 20240928 added
int set_interface_attribs(int fd, int speed)
{
    struct termios tty;

    if (tcgetattr(fd, &tty) < 0) {
        printf("Error from tcgetattr: %s¥n", strerror(errno));
        return -1;
    }

    cfsetospeed(&tty, (speed_t)speed);
    cfsetispeed(&tty, (speed_t)speed);

    tty.c_cflag |= (CLOCAL | CREAD);       /* ignore modem controls */
    tty.c_cflag &= ~CSIZE;
    tty.c_cflag |= CS8;                /* 8-bit characters */
    tty.c_cflag &= ~PARENB;        /* no parity bit */
    tty.c_cflag &= ~CSTOPB;        /* only need 1 stop bit */
    tty.c_cflag &= ~CRTSCTS;       /* no hardware flowcontrol */
```

```
	/* setup for non-canonical mode */
	tty.c_iflag &= ~(IGNBRK | BRKINT | PARMRK | ISTRIP | INLCR | IGNCR | ICRNL
| IXON);
	tty.c_lflag &= ~(ECHO | ECHONL | ICANON | ISIG | IEXTEN);
	tty.c_oflag &= ~OPOST;

	/* fetch bytes as they become available */
	tty.c_cc[VMIN] = 1;
	tty.c_cc[VTIME] = 1;

	if (tcsetattr(fd, TCSANOW, &tty) != 0) {
		printf("Error from tcsetattr: %s¥n", strerror(errno));
		return -1;
	}
	return 0;
}

// shinji-y 20240928 added
void set_mincount(int fd, int mcount)
{
	struct termios tty;

	if (tcgetattr(fd, &tty) < 0) {
		printf("Error tcgetattr: %s¥n", strerror(errno));
		return;
	}

	tty.c_cc[VMIN] = mcount ? 1 : 0;
	tty.c_cc[VTIME] = 5;		/* half second timer */

	if (tcsetattr(fd, TCSANOW, &tty) < 0)
		printf("Error tcsetattr: %s¥n", strerror(errno));
}



// shinji-y 20240928 added
```

```
int set_flowcontrol(int fd, int control)
{
	struct termios tty;
	memset(&tty, 0, sizeof tty);
	if (tcgetattr(fd, &tty) != 0)
	{
		perror("error from tggetattr");
		return -1;
	}

	if(control) tty.c_cflag |= CRTSCTS;
	else tty.c_cflag &= ~CRTSCTS;

	if (tcsetattr(fd, TCSANOW, &tty) != 0)
	{
		perror("error setting term attributes");
		return -1;
	}
	return 0;
}
```

改造モジュール：改造部分のみ抜粋：

```
/* shinji-y 20240928 Update a field in the GtkSingleSat view. */
static void update_field(GtkSingleSat * ssat, guint i)
{
	case SINGLE_SAT_FIELD_AZ:
		buff = g_strdup_printf("%3.2f¥302¥260", sat->az);
				sprintf(mybuff,"tio /dev/ttyACM0 ¥n",buff);			//  debug
shinji-y
// debug				system(mybuff);
		// debug shinji-y
				sprintf(mybuff,"¥"az:%s¥"",buff);		// debug shinji-y
				// debug system("tio /dev/ttyACM0");
// debug				system(mybuff);
		// debug shinji-y
				sprintf(mybuff,"az:%s¥n",buff);		// debug shinji-y
```

```
/*
                        sprintf(mybuff,"echo -e ¥"%s¥" > /dev/ttyACM0¥n",buff);
        // debug shinji-y
                                system(mybuff);
        // debug shinji-y
*/
if(shinjiyflag) {
    fd = open(portname, O_RDWR | O_NOCTTY | O_SYNC);
    if (fd < 0) {
        printf("Error opening %s: %s¥n", portname, strerror(errno));
        return -1;
    }
    /*baudrate 115200, 8 bits, no parity, 1 stop bit */
    set_interface_attribs(fd, B115200);
    //set_mincount(fd, 0);                  /* set to pure timed read */

    /* simple output */

                                    len = strlen(mybuff);
        write(fd, mybuff, len-3);

/*
    wlen = write(fd, xstr, xlen);
    if (wlen != xlen) {
        printf("Error from write: %d, %d¥n", wlen, errno);
    }
*/
    tcdrain(fd);    /* delay for output */

rc = set_flowcontrol(fd, 0);
if (rc != 0)
{
    perror("error setting flowcontrol: ");
    exit(-1);
}
```

```
rc = close(fd);
if (rc != 0)
{
      perror("error closing fd: ");
      exit(-1);
}

// debug 20240928       close(fd);
        shinjiyflag = 0;
}else {
                                    int serial_port = open("/dev/ttyACM0", O_WRONLY);


                                          // Check for errors
                                          if (serial_port < 0) {
                                               printf("Error   %i   from   open:   %s¥n",   errno,
strerror(errno));
                                          }



                                    // Create new termios struct, we call it 'tty' for convention
                                    struct termios tty;

                                    // Read in existing settings, and handle any error
                                    if(tcgetattr(serial_port, &tty) != 0) {
                                         printf("Error   %i   from   tcgetattr:   %s¥n",   errno,
strerror(errno));
                                         return 1;
                                    }

                                    tty.c_cflag &= ~PARENB; // Clear parity bit, disabling parity
(most common)
                                    tty.c_cflag &= ~CSTOPB; // Clear stop field, only one stop bit
used in communication (most common)
                                    tty.c_cflag &= ~CSIZE; // Clear all bits that set the data size
```

```
                tty.c_cflag |= CS8; // 8 bits per byte (most common)
                tty.c_cflag &= ~CRTSCTS; // Disable RTS/CTS hardware flow control (most common)
                tty.c_cflag |= CREAD | CLOCAL; // Turn on READ & ignore ctrl lines (CLOCAL = 1)

                tty.c_lflag &= ~ICANON;
                tty.c_lflag &= ECHO; // Disable echo shinji-y debug ~ECHO;
                tty.c_lflag &= ~ECHOE; // Disable erasure
                tty.c_lflag &= ~ECHONL; // Disable new-line echo
                tty.c_lflag &= ~ISIG; // Disable interpretation of INTR, QUIT and SUSP
                tty.c_iflag &= ~(IXON | IXOFF | IXANY); // Turn off s/w flow ctrl
                tty.c_iflag &= ~(IGNBRK|BRKINT|PARMRK|ISTRIP|INLCR|IGNCR|ICRNL); // Disable any special handling of received bytes

                tty.c_oflag &= ~OPOST; // Prevent special interpretation of output bytes (e.g. newline chars)
                tty.c_oflag &= ~ONLCR; // Prevent conversion of newline to carriage return/line feed
                // tty.c_oflag &= ~OXTABS; // Prevent conversion of tabs to spaces (NOT PRESENT ON LINUX)
                // tty.c_oflag &= ~ONOEOT; // Prevent removal of C-d chars (0x004) in output (NOT PRESENT ON LINUX)

                tty.c_cc[VTIME] = 10;      // Wait for up to 1s (10 deciseconds), returning as soon as any data is received.
                tty.c_cc[VMIN] = 0;

                // Set in/out baud rate to be 115200
                cfsetispeed(&tty, B115200);
                cfsetospeed(&tty, B115200);

                // Save tty settings, also checking for error
```

```
                                if (tcsetattr(serial_port, TCSANOW, &tty) != 0) {
                                                                printf("Error       %i       from
tcsetattr: %s¥n", errno, strerror(errno));
                                                                return 1;
                                        }



                                        // Write to serial port
                                        unsigned char msg[] = "Hello";
                                        int len;
                                        len = strlen(mybuff);
            write(serial_port, mybuff, len-1);

rc = set_flowcontrol(fd, 0);
if (rc != 0)
{
      perror("error setting flowcontrol: ");
      exit(-1);
}

rc = close(fd);
if (rc != 0)
{
      perror("error closing fd: ");
      exit(-1);
}


// debug 20240928                                       close(serial_port);
};
            break;
      case SINGLE_SAT_FIELD_EL:
            buff = g_strdup_printf("%3.2f¥302¥260", sat->el);
             sprintf(mybuff,"echo -e ¥"%s¥" > /dev/ttyACM0¥n",buff);                  //   debug
shinji-y
```

```
                    sprintf(mybuff,"el:%s¥n",buff);                // debug shinji-y
/*
                    sprintf(mybuff,"echo -e ¥"%s¥" > /dev/ttyACM0¥n",buff);
        // debug shinji-y
                            system(mybuff);
        // debug shinji-y
*/
                            serial_port = open("/dev/ttyACM0", O_WRONLY);

                                // Check for errors
                                if (serial_port < 0) {
                                    printf("Error %i from open: %s¥n", errno,
strerror(errno));
                                }



                            // Create new termios struct, we call it 'tty' for convention
// debug                    struct termios tty;

                            // Read in existing settings, and handle any error
                            if(tcgetattr(serial_port, &tty) != 0) {
                                printf("Error %i from tcgetattr: %s¥n", errno,
strerror(errno));
                                return 1;
                            }

                            tty.c_cflag &= ~PARENB; // Clear parity bit, disabling parity
(most common)
                            tty.c_cflag &= ~CSTOPB; // Clear stop field, only one stop bit
used in communication (most common)
                            tty.c_cflag &= ~CSIZE; // Clear all bits that set the data size
                            tty.c_cflag |= CS8; // 8 bits per byte (most common)
                            tty.c_cflag &= ~CRTSCTS; // Disable RTS/CTS hardware
flow control (most common)
                            tty.c_cflag |= CREAD | CLOCAL; // Turn on READ & ignore
```

ctrl lines (CLOCAL = 1)

```
                tty.c_lflag &= ~ICANON;
                tty.c_lflag &= ECHO; // Disable echo shinji-y debug ~ECHO;
                tty.c_lflag &= ~ECHOE; // Disable erasure
                tty.c_lflag &= ~ECHONL; // Disable new-line echo
                tty.c_lflag &= ~ISIG; // Disable interpretation of INTR, QUIT and SUSP
                tty.c_iflag &= ~(IXON | IXOFF | IXANY); // Turn off s/w flow ctrl
                tty.c_iflag &= ~(IGNBRK|BRKINT|PARMRK|ISTRIP|INLCR|IGNCR|ICRNL); // Disable any special handling of received bytes

                tty.c_oflag &= ~OPOST; // Prevent special interpretation of output bytes (e.g. newline chars)
                tty.c_oflag &= ~ONLCR; // Prevent conversion of newline to carriage return/line feed
                // tty.c_oflag &= ~OXTABS; // Prevent conversion of tabs to spaces (NOT PRESENT ON LINUX)
                // tty.c_oflag &= ~ONOEOT; // Prevent removal of C-d chars (0x004) in output (NOT PRESENT ON LINUX)

                tty.c_cc[VTIME] = 10;      // Wait for up to 1s (10 deciseconds), returning as soon as any data is received.
                tty.c_cc[VMIN] = 0;

                // Set in/out baud rate to be 115200
                cfsetispeed(&tty, B115200);
                cfsetospeed(&tty, B115200);

                // Save tty settings, also checking for error
                if (tcsetattr(serial_port, TCSANOW, &tty) != 0) {
                                printf("Error %i from tcsetattr: %s¥n", errno, strerror(errno));
                                return 1;
```

```
}


                                        // Write to serial port
// debug                                unsigned char msg[] = "Hello";
// debug                                int len;
                                        len = strlen(mybuff);
            write(serial_port, mybuff, len-1);

rc = set_flowcontrol(fd, 0);
if (rc != 0)
{
    perror("error setting flowcontrol: ");
    exit(-1);
}

rc = close(fd);
if (rc != 0)
{
    perror("error closing fd: ");
    exit(-1);
}


// debug 20240928                                   close(serial_port);

        break;
```

７．謝意

この素晴らしいオープンソースを提供そして公開してくれた OZ9AEC / Alexandru Csete 氏に深く感謝申し上げます。