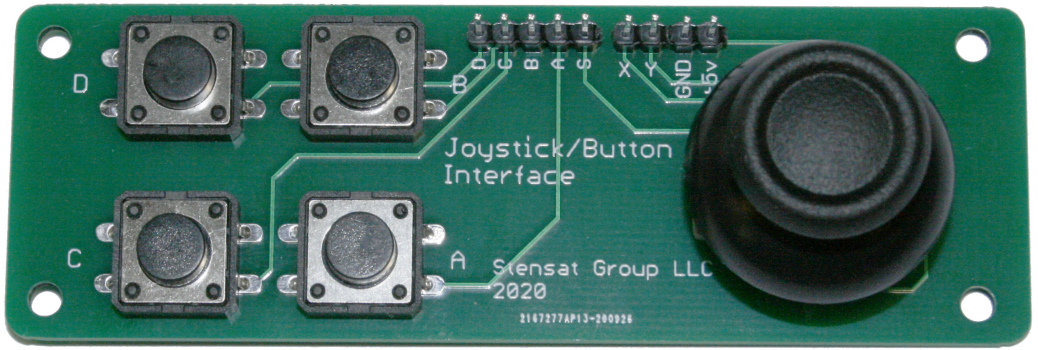
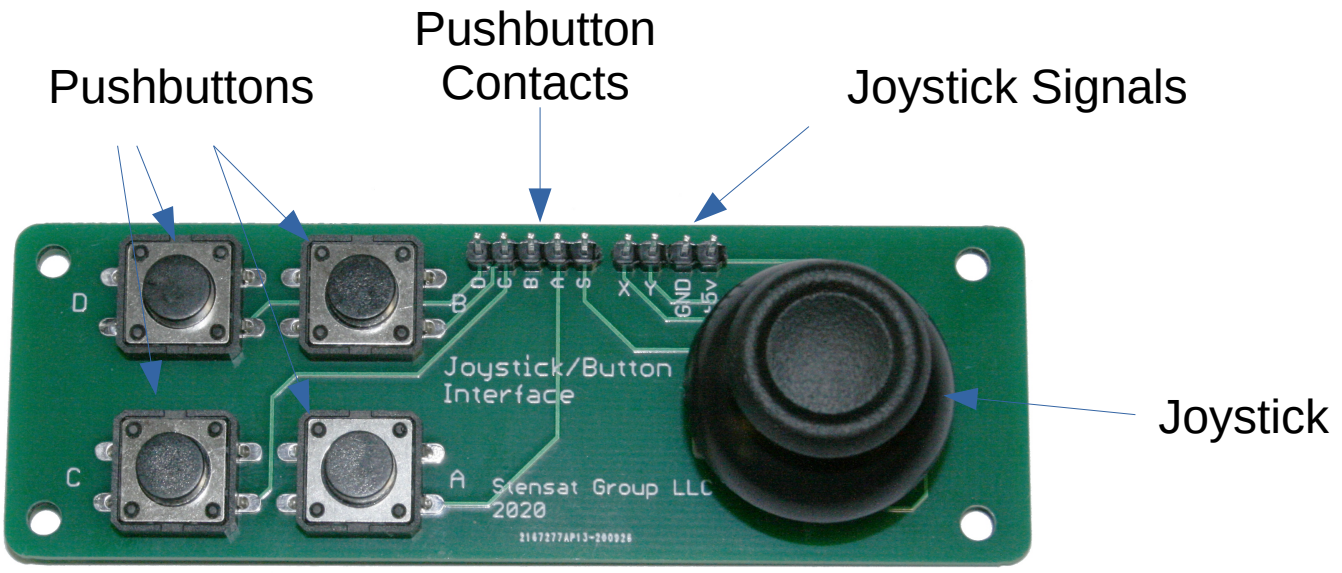
Joystick Interface



Joystick Module

- The joystick module consists of four tactile push buttons and an analog joystick that includes a button activated by pressing the joystick knob.



Joystick Module

- The push buttons are connected to the ground pin on one side and connected to A, B, C, D and S pins. When the push button is pressed, the pin A, B, C, D, or S are connected to ground through the button.
- In order to use the buttons, a pull-up resistor needs to be included to pull the pin to 3.3 volts when the button is not pushed. The SLATE processor has a pull-up resistor built in for each digital pin. It is activated using the **pinMode()** function.
  - Example: **pinMode(5,INPUT_PULLUP);**
- The following digital pins can be used with the push buttons.
  - D0, D2, D4, D5, D12, D13, D14.

Joystick Module

- Connect the following digital pins to push buttons
  - A to D12
  - B to D13
  - C to D14
  - D to D0
  - Connect GND on the joystick module to any of GND pin on the SLATE.
- Load the Arduino program and run it and open the serial monitor.
- With none of the push buttons pressed, the serial monitor should display four ones.
- Press any of the buttons and observe the value change to zero.

```
void setup() {
  Serial.begin(115200);
  pinMode(12,INPUT_PULLUP);
  pinMode(13,INPUT_PULLUP);
  pinMode(14,INPUT_PULLUP);
  pinMode(0,INPUT_PULLUP);
}

void loop() {
  char buf[32];
  int a = digitalRead(12);
  int b = digitalRead(13);
  int c = digitalRead(14);
  int d = digitalRead(0);
  snprintf(buf,32,"%d %d %d %d",a,b,c,d);
  Serial.println(buf);
  delay(20);
}
```

Joystick Module

- With none of the push buttons pressed, the serial monitor should display four ones.
- Press any of the buttons and observe the value change to zero.
- Any combination of buttons can be pressed simultaneously.

```
void setup() {
  Serial.begin(115200);
  pinMode(12,INPUT_PULLUP);
  pinMode(13,INPUT_PULLUP);
  pinMode(14,INPUT_PULLUP);
  pinMode(0,INPUT_PULLUP);
}

void loop() {
  char buf[32];
  int a = digitalRead(12);
  int b = digitalRead(13);
  int c = digitalRead(14);
  int d = digitalRead(0);
  snprintf(buf,32,"%d %d %d %d",a,b,c,d);
  Serial.println(buf);
  delay(20);
}
```

Joystick Module

- This python program shows how to use the buttons to maneuver a square in a canvas using Tkinter.
- The serial interface is used to capture the button states from the program on the previous page. The serial interface is opened with a timeout parameter set to .01 seconds.
- A canvas of 1000 pixels by 800 pixels is created with a black background.
- A rectangle is created for the canvas and is assigned a variable **veh**.

```
from tkinter import *
import serial
import threading

s = serial.Serial('COM4',115200,timeout=.01);

top = Tk()
c = Canvas(top,height=800,width=1000,bg='black')
veh = c.create_rectangle(500,400,550,450,fill='red')
```

Joystick Module

- This function will be executed as a thread.
- velx and vely variables set the speed of the rectangle. posx and posy set the position of the rectangle.
- After initializing the variables, the function goes into an infinite loop. This is required for the function to continuously run as a thread.

```
def check_buttons():
    velx = 0
    vely = 0
    posx = 500
    posy = 400
    while 1:
        dat = s.readline()
        if dat != None:
            dat = dat.decode('utf-8')
            but = dat.split(" ")
            if len(but) == 4:
                if but[0] == '0':
                    velx = velx + .1
                if but[1] == '0':
                    velx = velx - .1
                if but[2] == '0':
                    vely = vely + .1
                if but[3] == '0\r\n':
                    vely = vely - .1
                print(but,velx,vely)
                posx = posx + velx
                posy = posy + vely
                if posx > 1000:
                    posx = 0
                elif posx < 0:
                    posx = 1000
                if posy > 800:
                    posy = 0
                elif posy < 0:
                    posy = 800
        c.coords(veh,int(posx),int(posy),int(posx)+50,int(posy)+50)
```

Joystick Module

- **dat** is assigned any serial data received. **s.readline()** waits for a line of text terminated by a linefeed to be received. This is where the timeout parameter is used. **s.readline()** will wait for a line of data to arrive. If timeout was defined, **s.readline()** will quit after the timeout and let the code continue executing. This is why the next line checks if **dat** contains any data. **if dat !=None** checks if **dat** is not empty and will execute the indented code if not empty.

```
def check_buttons():
    velx = 0
    vely = 0
    posx = 500
    posy = 400
    while 1:
        dat = s.readline()
        if dat != None:
            dat = dat.decode('utf-8')
            but = dat.split(" ")
            if len(but) == 4:
                if but[0] == '0':
                    velx = velx + .1
                if but[1] == '0':
                    velx = velx - .1
                if but[2] == '0':
                    vely = vely + .1
                if but[3] == '0\r\n':
                    vely = vely - .1
                print(but,velx,vely)
                posx = posx + velx
                posy = posy + vely
                if posx > 1000:
                    posx = 0
                elif posx < 0:
                    posx = 1000
                if posy > 800:
                    posy = 0
                elif posy < 0:
                    posy = 800
        c.coords(veh,int(posx),int(posy),int(posx)+50,int(posy)+50)
```

Joystick Module

- Next, **dat** is converted to a string with the **dat.decode()** function.
- **but** is assigned a list of string values from the **dat.split()** function.
- Next, **but** variable is checked to have four elements. If there are four elements, then each one is checked and the velocity in the selected direction is increased.
- Notice **but[3] == '0\r\n'**. This is because **Serial.println()** adds carriage return and line feed characters to the end of the text.

```
def check_buttons():
    velx = 0
    vely = 0
    posx = 500
    posy = 400
    while 1:
        dat = s.readline()
        if dat != None:
            dat = dat.decode('utf-8')
            but = dat.split(" ")
            if len(but) == 4:
                if but[0] == '0':
                    velx = velx + .1
                if but[1] == '0':
                    velx = velx - .1
                if but[2] == '0':
                    vely = vely + .1
                if but[3] == '0\r\n':
                    vely = vely - .1
                print(but,velx,vely)
                posx = posx + velx
                posy = posy + vely
                if posx > 1000:
                    posx = 0
                elif posx < 0:
                    posx = 1000
                if posy > 800:
                    posy = 0
                elif posy < 0:
                    posy = 800
        c.coords(veh,int(posx),int(posy),int(posx)+50,int(posy)+50)
```

Joystick Module

- The position is updated based on velocity.
- The positions are checked for going off the canvas and changed to the opposite side of the canvas.
- Lastly, the coordinates of the rectangle are updated.

```
def check_buttons():
    velx = 0
    vely = 0
    posx = 500
    posy = 400
    while 1:
        dat = s.readline()
        if dat != None:
            dat = dat.decode('utf-8')
            but = dat.split(" ")
            if len(but) == 4:
                if but[0] == '0':
                    velx = velx + .1
                if but[1] == '0':
                    velx = velx - .1
                if but[2] == '0':
                    vely = vely + .1
                if but[3] == '0\n':
                    vely = vely - .1
                print(but,velx,vely)
                posx = posx + velx
                posy = posy + vely
                if posx > 1000:
                    posx = 0
                elif posx < 0:
                    posx = 1000
                if posy > 800:
                    posy = 0
                elif posy < 0:
                    posy = 800
        c.coords(veh,int(posx),int(posy),int(posx)+50,int(posy)+50)
```

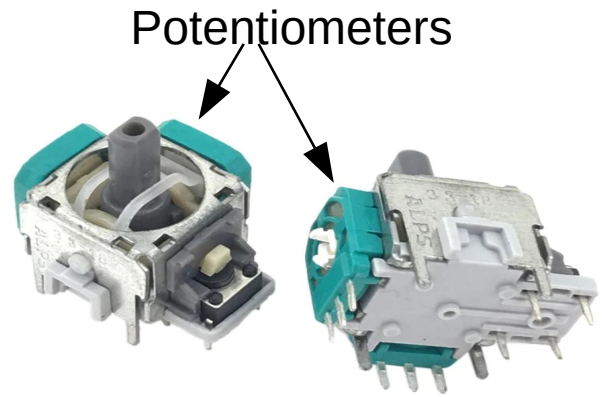
Joystick Module

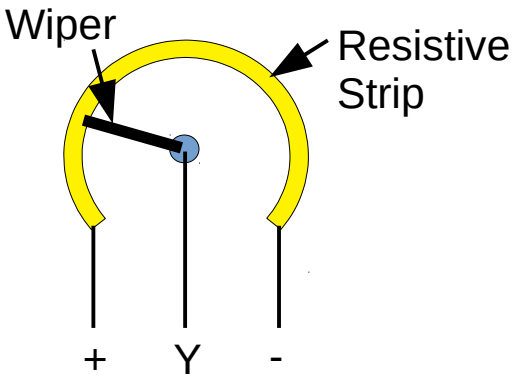
- The last part of the program sets up the thread and starts the thread called **check_buttons**.
- Finally, the canvas is packed into the window and the **mainloop()** is started.
- Run the previous Arduino program. Run this program and try out the buttons and see how the rectangle reacts.
- You can turn this into a game.

```
x = threading.Thread(target=check_buttons)
x.start()
c.pack()
top.mainloop()
```

Joystick Module

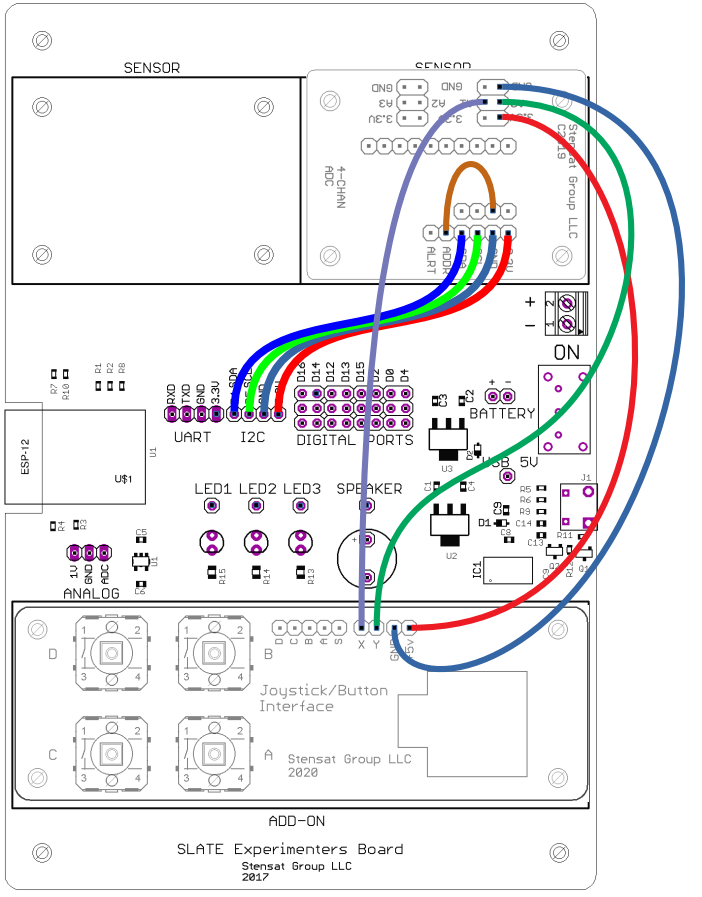
- The joystick is built with two potentiometers providing analog outputs for the two axis.
- The potentiometer consists of a resistive material. The ends of the material have pins connected. The resistance is 10,000 ohms between the two pins.
- A wiper is connected to a third pin. The wiper moves across the resistive material and have a changing resistance relative to the other pins.
- With a voltage applied to the ends of the resistive material, the voltage from the wiper will vary from 0 to the input voltage. The closer it gets to one end, the closer it is at that voltage level.





Joystick Module

- Since the SLATE has only one analog port and the joystick has two analog outputs, the ADC module will be needed to capture both axis.
- Leave the buttons wired as is. They will be used later.
- Connect the ADC to the SLATE I2C port.
- Connect the joystick to the ADC as shown.
  - 5V to 3.3V on the ADC board
  - GND to the GND on the ADC board
  - Y to A0 on the ADC board
  - X to A1 on the ADC board



Joystick Module

- Start a new Arduino program.
- First, the function to read the ADC values from the ADC module needs to be inserted into the program. This is taken from the ADC lesson.
- This function will measure the voltage of the selected channel.

```
#include <Wire.h>

#define ADC 0x48

short readADC(unsigned char addr) {
  unsigned char b;
  b = (addr << 4) | 0xc3;
  Wire.beginTransmission(ADC);
  Wire.write(0x01);
  Wire.write(b);
  Wire.write(0x83);
  Wire.endTransmission();
  delay(10);
  Wire.beginTransmission(ADC);
  Wire.write(0x00);
  Wire.endTransmission(false);
  Wire.requestFrom(ADC,2);
  while(Wire.available() < 2) delay(1);
  int c = Wire.read();
  int d = Wire.read();
  short res = (c << 8) + b;
  return res;
}
```

Joystick Module

- This section of the code is similar to the ADC module lesson.
- The ADC values are printed and not converted to voltages.
- Load and run the program and observe the range of the ADC when moving the joystick in the serial monitor.

```
void setup() {
  Serial.begin(115200);
  Wire.begin(4,5);
}

void loop() {
  short x = readADC(0);
  short y = readADC(1);
  Serial.print(x);
  Serial.print(" ");
  Serial.println(y);
  delay(100);
}
```

Joystick Module

- Let's combine the buttons and joystick into one Arduino program. Keep the **readADC()** function from the last program.
- The digital pins need to be configured as inputs with pull-up resistors enabled.
- The **loop()** collects the analog voltages and reads the state of each button.
- All the information is put in a string comma separated and sent out the serial interface.

```
void setup() {
  Serial.begin(115200);
  Wire.begin(4,5);
  pinMode(12,INPUT_PULLUP);
  pinMode(13,INPUT_PULLUP);
  pinMode(14,INPUT_PULLUP);
  pinMode(0,INPUT_PULLUP);
}

void loop() {
  char buf[32];
  short x = readADC(0);
  short y = readADC(1);
  int a = digitalRead(12);
  int b = digitalRead(13);
  int c = digitalRead(14);
  int d = digitalRead(0);
  snprintf(buf,32,"%d,%d,%d,%d,%d,%d",x,y,a,b,c,d);
  Serial.println(buf);
  delay(50);
}
```

Joystick Module

- For the python program, the program will be similar to the one using the buttons except the joystick will be used to move the square around. With the analog joystick, you can control the acceleration of the square.
- One of the things that needs to be done is calibrate the center position of the joystick. No two joysticks will have the same center value.
- The buttons will be used to change the color of the square.

Joystick Module

- The beginning of the program includes the needed modules, the serial interface, threading and tkinter.
- The next two lines define scale factors. They will be explained later.
- The serial port is opened, Tk() is created along with the canvas, the square and text stating the joystick is being calibrated. The text will appear when the program starts and the window opens.

```
import serial
import threading
from tkinter import *

SCALE = 1000
SCALE_VEL = 10

s = serial.Serial('/dev/cu.usbserial-DN02SIRH',115200)
top = Tk()
c = Canvas(top,height=800,width=1000,bg='black')
veh = c.create_rectangle(500,400,550,450,fill='red')
msg = c.create_text(500,100,text="Calibrating, do not touch joystick",fill='yellow')
```

Joystick Module

- This function will be run as a thread in parallel with the tkinter mainloop() function.
- Just like in the previous python program, variables are initialized. Two new variables are added. These will be used to subtract the center position from the joystick measurements. This will allow for negative values when the joystick axis are moved left and down.

```
def check_joystick():
    velx = 0
    vely = 0
    posx = 500
    posy = 400
    ctrx = 0
    ctry = 0
    for i in range(0,100):
        dat = s.readline()
        if dat != None:
            dat = dat.decode('utf-8',ignore)
            joy = dat.split(',')
            if len(joy) == 6:
                ctrx = ctrx + int(joy[1])
                ctry = ctry + int(joy[0])
    ctrx = ctrx / 100
    ctry = ctry / 100
    print(ctrx,ctry)
    c.delete(msg)
```

Joystick Module

- This for loop samples the joystick 100 times and accumulates the measurents in the variables ctrx and ctry.
- After the for loop, the variables are divided by 100 with the results being the average center position of the joystick. During the for loop, the joystick should not be touched.
- At the end of this section of code, the calibration message is deleted and removed from the display.

```
def check_joystick():
    velx = 0
    vely = 0
    posx = 500
    posy = 400
    ctrx = 0
    ctry = 0
    for i in range(0,100):
        dat = s.readline()
        if dat != None:
            dat = dat.decode('utf-8',ignore)
            joy = dat.split(',')
            if len(joy) == 6:
                ctrx = ctrx + int(joy[1])
                ctry = ctry + int(joy[0])
    ctrx = ctrx / 100
    ctry = ctry / 100
    print(ctrx,ctry)
    c.delete(msg)
```

Joystick Module

- Continuing in the function, the thread goes into a while loop that will run forever.
- Again, the joystick and button data is read and processed as before.
- The final joystick values are calculated by subtracting the center position from the measured value. The result is then scaled by the SCALE value. Since the joystick measurements range from 0 to 32000+, the results need to be scaled for calculating the velocities.

```
def check_joystick():
    velx = 0
    vely = 0
    posx = 500
    posy = 400
    ctrx = 0
    ctry = 0
    for i in range(0,100):
        dat = s.readline()
        if dat != None:
            dat = dat.decode('utf-8',ignore)
            joy = dat.split(',')
            if len(joy) == 6:
                ctrx = ctrx + int(joy[1])
                ctry = ctry + int(joy[0])
    ctrx = ctrx / 100
    ctry = ctry / 100
    print(ctrx,ctry)
    c.delete(msg)
    while 1:
        dat = s.readline()
        if dat != None:
            dat = dat.decode('utf-8',ignore)
            joy = dat.split(',')
            joyx = (int(joy[1]) - ctrx) / SCALE
            joyy = -(int(joy[0]) - ctry) / SCALE
            if abs(joyx) < .2:
                joyx = 0
            if abs(joyy) < .2:
                joyy = 0
```

Joystick Module

- And since the calibration is not perfect, there will be a residual offset. This part of the code sets a threshold of when to use joyx and joyy to calculate velocities. If the values are below .2, the variables are set to 0. This eliminates any slow drifting from occurring when the joystick is left at center.

```
def check_joystick():
    velx = 0
    vely = 0
    posx = 500
    posy = 400
    ctrx = 0
    ctry = 0
    for i in range(0,100):
        dat = s.readline()
        if dat != None:
            dat = dat.decode('utf-8',ignore)
            joy = dat.split(',')
            if len(joy) == 6:
                ctrx = ctrx + int(joy[1])
                ctry = ctry + int(joy[0])
    ctrx = ctrx / 100
    ctry = ctry / 100
    print(ctrx,ctry)
    c.delete(msg)
    while 1:
        dat = s.readline()
        if dat != None:
            dat = dat.decode('utf-8',ignore)
            joy = dat.split(',')
            joyx = (int(joy[1]) - ctrx) / SCALE
            joyy = -(int(joy[0]) - ctry) / SCALE
            if abs(joyx) < .2:
                joyx = 0
            if abs(joyy) < .2:
                joyy = 0
```

Joystick Module

- Continuing in the function, the velocity is calculated and another scaling factor is included. This controls the acceleration rate.
- The position calculation and window edge detection is the same as the previous program.

```
        if abs(joyx) < .2:
            joyx = 0
        if abs(joyy) < .2:
            joyy = 0
        velx = velx + joyx / SCALE_VEL
        vely = vely + joyy / SCALE_VEL
        posx = posx + velx
        posy = posy + vely
        if posx > 1000:
           posx = 0
        elif posx < 0:
           posx = 1000
        if posy > 800:
           posy = 0
        elif posy < 0:
           posy = 800
        if joy[2] == '0':
            c.itemconfig(veh,fill='blue')
        if joy[3] == '0':
            c.itemconfig(veh,fill='red')
        if joy[4] == '0':
            c.itemconfig(veh,fill='green')
        if joy[5] == '0\r\n':
            c.itemconfig(veh,fill='yellow')
    c.coords(veh,int(posx),int(posy),int(posx)+50,int(posy)+50)
```

Joystick Module

- This portion of the code checks the state of the buttons and changes the color of the square.
- Lastly, the position of the square is updated.
- This is the end of the function.

```
        if abs(joyx) < .2:
            joyx = 0
        if abs(joyy) < .2:
            joyy = 0
        velx = velx + joyx / SCALE_VEL
        vely = vely + joyy / SCALE_VEL
        posx = posx + velx
        posy = posy + vely
        if posx > 1000:
           posx = 0
        elif posx < 0:
           posx = 1000
        if posy > 800:
           posy = 0
        elif posy < 0:
           posy = 800
        if joy[2] == '0':
            c.itemconfig(veh,fill='blue')
        if joy[3] == '0':
            c.itemconfig(veh,fill='red')
        if joy[4] == '0':
            c.itemconfig(veh,fill='green')
        if joy[5] == '0\r\n':
            c.itemconfig(veh,fill='yellow')
    c.coords(veh,int(posx),int(posy),int(posx)+50,int(posy)+50)
```

Joystick Module

- At the end of the program, the thread is created and started.
- The canvas is packed into the window and the tkinter mainloop() is started.
- Load and start the arduino program and start this program.
- Do not touch the joystick while the calibration message is visible. Once the message disappears, start playing with the joystick and buttons.

```
            if abs(joyx) < .2:
                joyx = 0
            if abs(joyy) < .2:
                joyy = 0
            velx = velx + joyx / SCALE_VEL
            vely = vely + joyy / SCALE_VEL
            posx = posx + velx
            posy = posy + vely
            if posx > 1000:
               posx = 0
            elif posx < 0:
               posx = 1000
            if posy > 800:
               posy = 0
            elif posy < 0:
               posy = 800
            if joy[2] == '0':
                c.itemconfig(veh,fill='blue')
            if joy[3] == '0':
                c.itemconfig(veh,fill='red')
            if joy[4] == '0':
                c.itemconfig(veh,fill='green')
            if joy[5] == '0\r\n':
                c.itemconfig(veh,fill='yellow')
        c.coords(veh,int(posx),int(posy),int(posx)+50,int(posy)+50)

x = threading.Thread(target=check_joystick)
x.start()
c.pack()
top.mainloop()
```