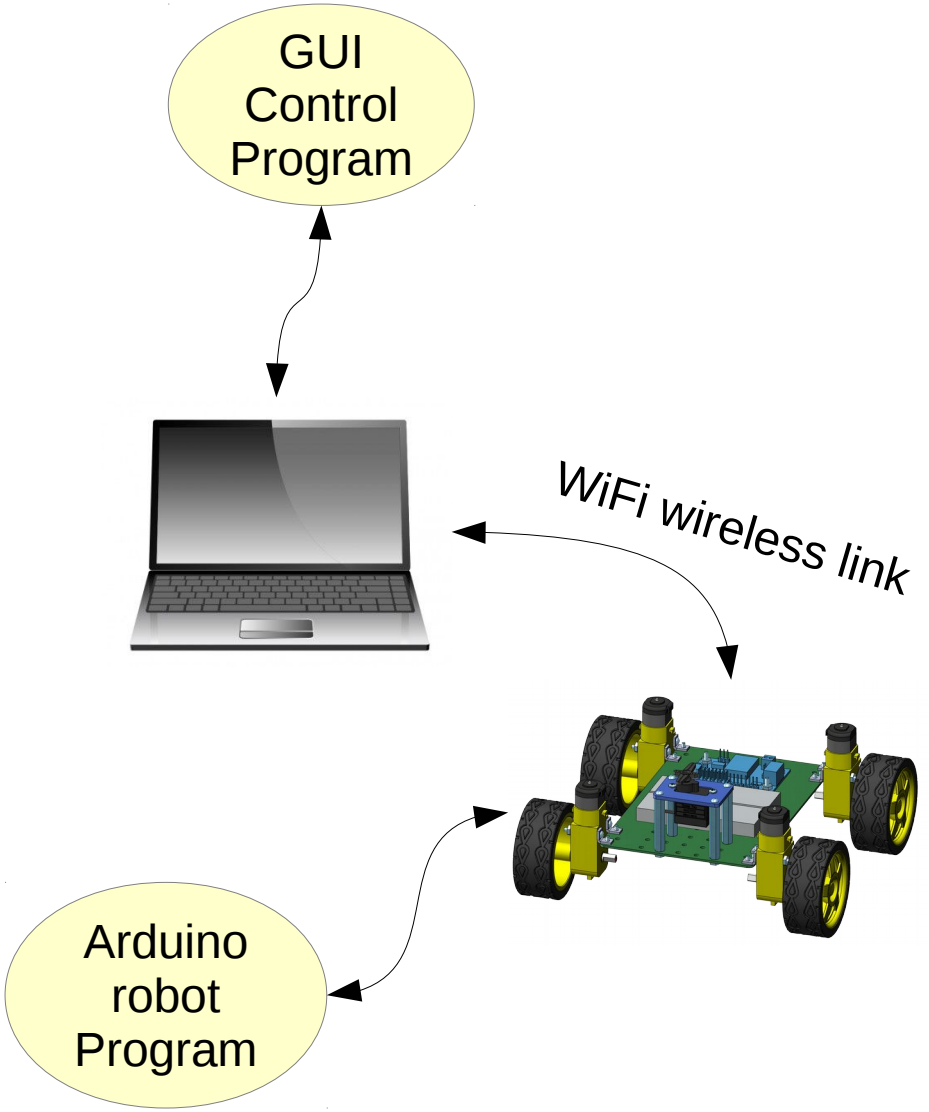
Building a GUI From Scratch

Processing Graphical User Interface

- In this lesson, you will learn how to create some simple GUI objects to control the robot. The GUI objects will be sliders and a joystick.
- GUI libraries could be used but most do not work as needed.
- For the robot, there is a robotic arm with three servos that need to be controlled. They will be controlled with sliders.
- For motion control, a joystick will be created to drive and steer the robot.

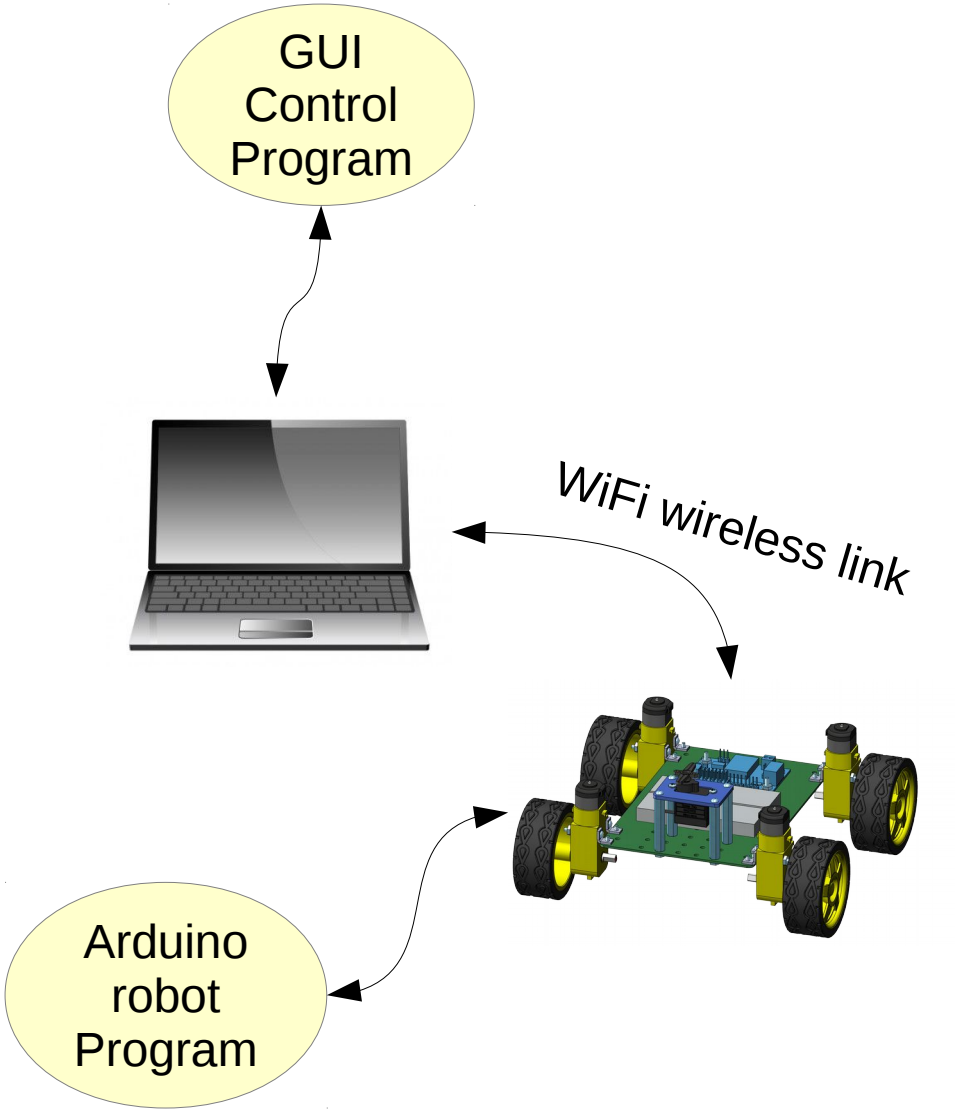
System Architecture

- A system architecture needs to be defined.
- The system architecture here consists of a laptop running a GUI control program with a WiFi interface and a robot with a robotic arm and a WiFi interface.
- The communication method will be wireless using WiFi.



System Architecture

- WiFi requires an access point and clients.
- The robot WiFi will act as an access point requiring the computer running the GUI control program to connect to the robot as a client.



System Protocol

- Now that the method of communications has been defined, how information is passed or structured needs to be defined.
- A protocol or command format needs to be defined. A simple protocol will be defined here. It will consist of four bytes. The first byte will be the header and is set to an arbitrary value of ASCII character 'C'. The rest of the bytes will be used to control the operation of the robot. The protocol is shown below.

Header	Left	Right	Base
'C'	Byte	Byte	Byte

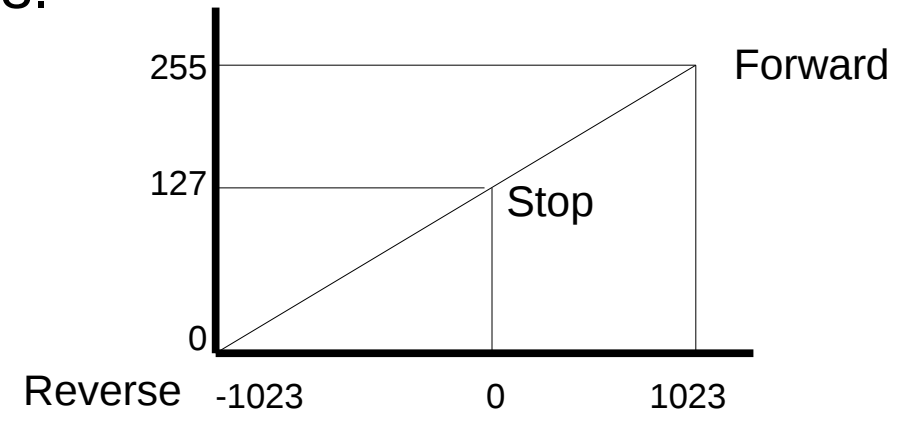
System Protocol

- The second byte is called **Left** and will control the direction and speed of the left wheel.
- The third byte is called **Right** and will control the direction and speed of the right wheel.
- The fourth byte is called **Base** and controls the position of the base servo on the robotic arm.
- The fifth byte is called **Arm** and controls the arm servo.
- The sixth byte is called **Elbow** and controls the elbow servo.

Header	Left	Right	Base
'C'	Byte	Byte	Byte

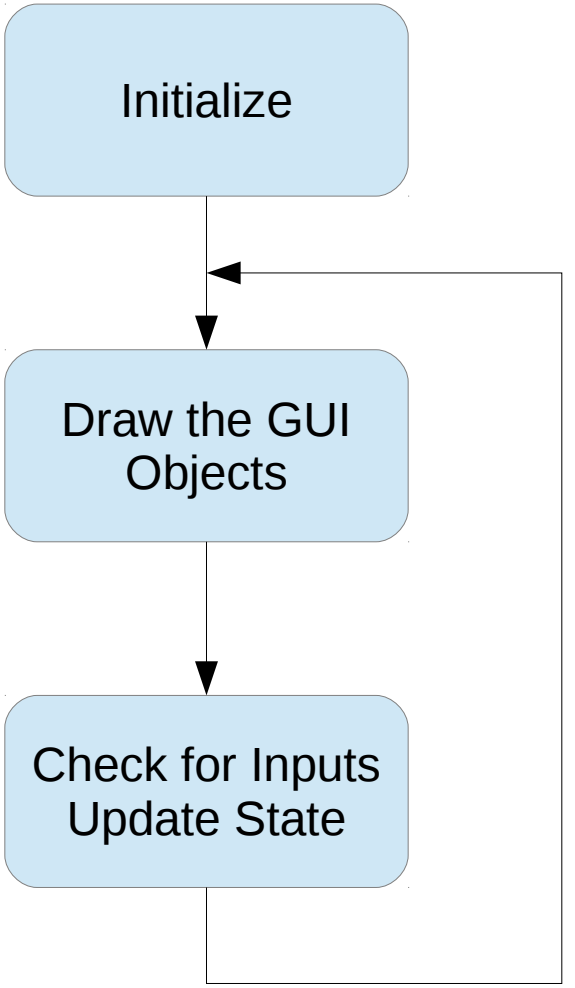
System Protocol

- The PWM value range for the motors is 0 to 1023. This is a 10 bit number which is larger than a byte (8 bits). Since fine speed control is really not needed, one byte will be used to control the speed and direction of the robot motors.
- A byte has a range of 0 to 255. The mid point is 127. The value 127 will be defined as the stop state for the motors.
- The range of 0 to 126 will be the reverse speed with 0 being the fastest and 126 the slowed. The range of 128 to 255 will be the forward speed with 255 being the fastest and 128 being the slowest.
- For the robot, the mapping and direction will be mapped from 0 to 255 to -1023 to 1023.



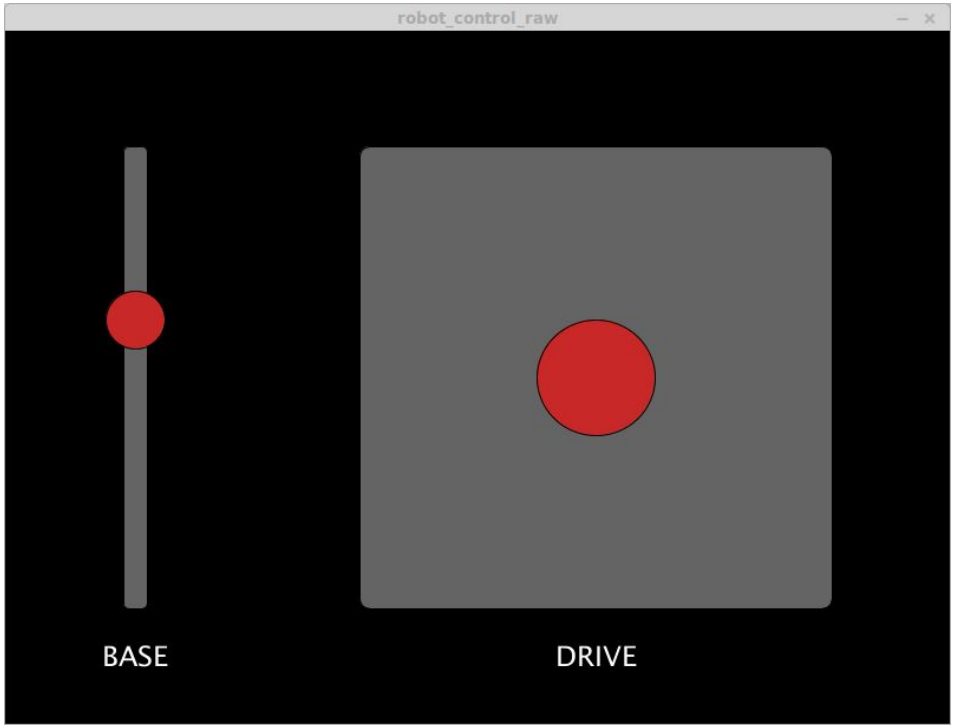
Design of the Program

- The program can be broken into three parts
  - Initialization
  - Drawing
  - Check for input and update the state of the robot



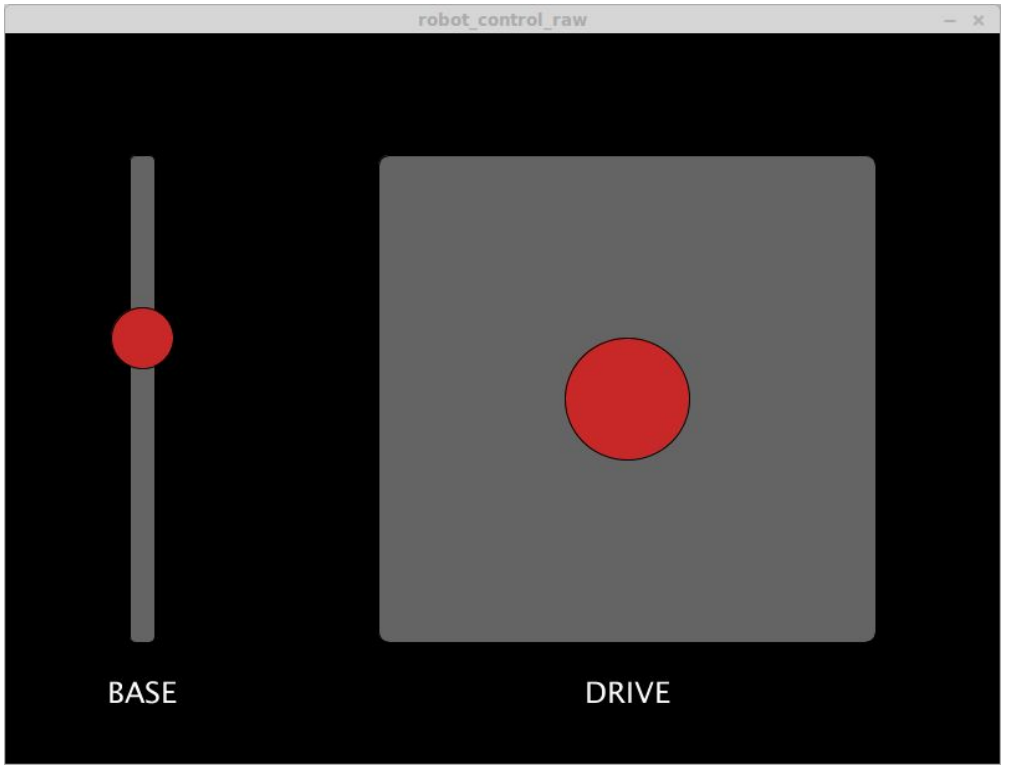
GUI Layout

- Three sliders need to be created to control the robotic arm and a joystick needs to be created for robot motion control.
- All the controls will be laid out horizontally.



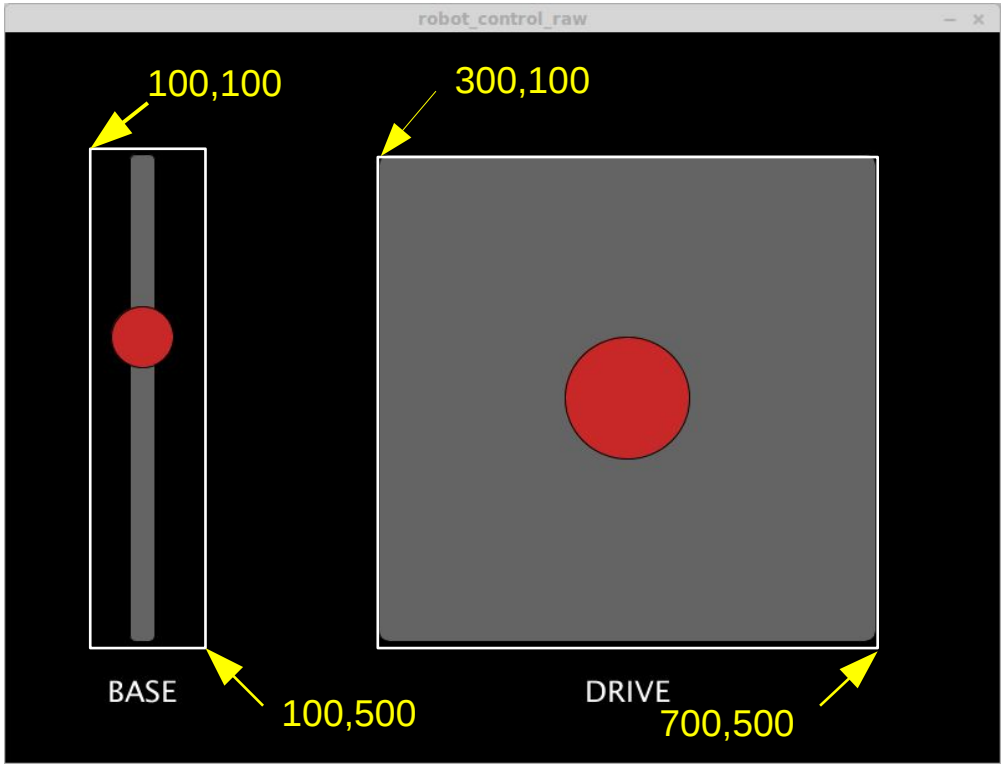
GUI Object Locations

- Next is creating a layout of the window. The window will be set to 800 by 600 pixels.
- The sliders will be 400 pixels tall and 100 pixels wide.
- The joystick will be 400 by 400 pixels.



GUI Object Locations

- The objects will be located with the window coordinates shown.
- The numbers indicate the top left corner and the bottom right corner. These values will be used as boundaries for each control to determine if the mouse is controlling the particular control.



Setting Up

- Slider positions need to be tracked and updated based on user input.
- Variables will be used to store the current position of the sliders and joystick.
- The **setup()** funtion is where the window is created and the position variables set.
- The text size for the labels is set to 24 point font.
- **textAlign()** will center text around the position specified.

```
int base;
int jx,jy;

void setup() {
  size(800,600);
  base = 250;
  jx = 500;
  jy = 300;
  textSize(24);
  textAlign(CENTER);
}
```

Drawing

- In the draw function, the GUI objects will be rendered. Since draw() is executed repeatedly, the objects will be redrawn repeatedly. This allows automatic update of the slider positions.
- First, the background slider and joystick objects are generated.

```
int base;
int jx,jy;

void setup() {
  size(1200,700);
  base = 300;
  jx = 500;
  jy = 300;
  textSize(24);
  textAlign(CENTER);
}

void draw() {
  background(0); // clear display
  fill(150);     // set color to grey
  rect(100,100,20,400,5);
  rect(300,100,400,400,10);
```

Drawing

- Labels are added to the GUI display. The positions are below the slider and joystick and centered.

setup() function not shown to save space

```
void draw() {
  fill(150);      // set color to grey
  rect(100,100,20,400,5);
  rect(300,100,400,400,10);  fill(255);
  text(“BASE”,110,550);
  text(“DIRECTION”,500,550);
```

Drawing

- The slider and joystick position controls are generated with ellipses.
- The Y position of the sliders are variables allowing them to be moved up and down.
- The joystick has variables for both X and Y position.
- Run the program to see the layout.

setup() function not shown to save space

```
void draw() {
  fill(150);      // set color to grey
  rect(100,100,20,400,5);
  rect(300,100,400,400,10);  fill(255);
  text("BASE",110,550);
  text("DIRECTION",500,550);
  ellipse(110,base,50,50);
  ellipse(jx,jy,100,100);
}
```

Drawing

- Next is to detect used input. The only input required is detecting when the mouse button is pressed and the location of the mouse. This also works for touch screens.
- When the mouse button is pressed, the four functions will be executed. Each function will test a specific slider or joystick.

```
void draw() {
  fill(150);      // set color to grey
  rect(100,100,20,400,5);
  rect(300,100,400,400,10);  fill(255);
  text("BASE",110,550);
  text("DIRECTION",500,550);
  fill(200,40,40);
  ellipse(110,base,50,50);
  ellipse(jx,jy,100,100);
  if(mousePressed) {
    check_base();
    check_joystick();
  }
}
```

Drawing

- Below the draw() function, add the two functions for checking the slider and joystick.
- To the right is the check_base() function. It checks if the mouse button is in the boundary area of the base slider. If it is, the base slider control position is updated to the current mouse Y position.

```
void check_base() {
  if((mouseX > 50) && (mouseX < 150)) {
    if((mouseY > 100) && (mouseY < 500)) {
      base = (int)mouseY;
    }
  }
}
```

Drawing

- To the right is the check_joystick() function. It checks if the mouse button is in the area of the joystick. If it is, the joystick control position is updated to the current mouse X and Y position.
- Run the program. The sliders and joystick controls should move.

```
void check_joystick() {
  if((mouseX > 300) && (mouseX < 700)) {
    if((mouseY > 100) && (mouseY < 500)) {
      jy = (int)mouseY;
      jx = (int)mouseX;
    }
  }
}
```

Drawing

- Notice the joystick position does not go back to center when the mouse button is released. Code has to be added to make that happen.
- The function **mouseReleased()** will be used to reposition the joystick control when the mouse button is released.
- Add the function to the end of the program.
- Run the program again and check the joystick.

```
void mouseReleased() {
  jx = 500;
  jy = 300;
}
```

Robot Control

- At this point, the GUI is built and operational. The code needs to be updated to control the robot via WiFi link.
- The network library needs to be added and new code needs to be added to portions of the program.
- For simplicity, the new code will be added after all the GUI control checks. The variables **base**, **arm**, **elbow**, **jx** and **jy** will be used to calculate servo position values and motor drive speed for the robot.

Adding Network Library

- Move the cursor to the top of the program.
- Include the network library.
- Add a network client object **c**.
- Add a byte array below it.
- In the **setup()** function, add the line to connect to the robot.

```
import processing.net.*;

Client c;
byte[] cmd = new byte[6];

int base;
int jx,jy;

void setup() {
  size(800,600);
  base = 250;
  jx = 500;
  jy = 300;
  textSize(24);
  textAlign(CENTER);
  c = new Client(this,"192.168.4.1",80);
}
```

Adding Network Library

- The byte array cmd will be the command bytes sent to the robot. It is a 6 byte command packet. The first byte is set to the letter C. This is the header of the packet.
- The next byte will be the left motor speed control
- The third byte will be the right motors speed control
- The fourth byte is the robotic arm base servo.
- The fifth byte is the robotic arm arm servo.
- The sixth byte is the robotic arm elbow servo.

```
import processing.net.*;

Client c;
byte[] cmd = new byte[6];

int base;
int jx,jy;

void setup() {
  size(800,600);
  base = 250;
  jx = 500;
  jy = 300;
  textSize(24);
  textAlign(CENTER);
  c = new Client(this,"192.168.4.1",80);
}
```

Adding Network Library

- The byte array needs to be initialized.
- The header is set to 'C'.
- The motors are set to 0.
- The servos are set to 179 degrees.
- This completes all changes to the **setup()** function.

```
void setup() {
  size(800,600);
  base = 250;
  jx = 500;
  jy = 300;
  textSize(24);
  textAlign(CENTER);
  c = new Client(this,"192.168.4.1",80);
  cmd[0] = 'C';          // header
  cmd[1] = (byte)0;      // left motor
  cmd[2] = (byte)0;      // right motor
  cmd[3] = (byte)90;     // base servo

}
```

Robot Control

- At the end of the **draw()** function, the servo angle calculations will be added. The **map()** function will be used to translate the slider control position to the servo angle. For the base servo, the angle will have a range of 1 to 179.
- The slider position range is 100 to 500.

```
void draw() {
  fill(150);      // set color to grey
  rect(100,100,20,400,5);
  rect(300,100,400,400,10);  fill(255);
  text("BASE",110,550);
  text("DIRECTION",500,550);
  fill(200,40,40);
  ellipse(110,base,50,50);
  ellipse(jx,jy,100,100);
  if(mousePressed) {
    check_base();
    check_joystick();
  }
  float angle1 = map(base,100,500,1,179);
}
```

Robot Control

- The joystick position is converted to motor speed and direction for the left and right motors.

```
void draw() {
  fill(150);      // set color to grey
  rect(100,100,20,400,5);
  rect(300,100,400,400,10);  fill(255);
  text("BASE",110,550);
  text("DIRECTION",500,550);
  fill(200,40,40);
  ellipse(110,base,50,50);
  ellipse(jx,jy,100,100);
  if(mousePressed) {
    check_base();
    check_joystick();
  }
  float angle1 = map(base,100,500,1,179);
  float drive = map(jy,100,500,1,-1);
  float steer = map(jx,300,700,-1,1);
  float left = (drive + steer) * 255.0;
  float right = (drive – steer) * 255.0;
  left= constrain(left, -255,255);
  right = constrain(right,-255,255);
  left = map(left,-255,255,0,255);
  right = map(right,-255,255,0,255);
}
```

Robot Control

- All the calculated angles and motor speed is then copied to the command packet. The command packet is made up of bytes so the variables are type casted which converts the values to byte size values.
- The **c.write()** function sends the command packet to the robot over the network connection.

```
Above code not shown to save space
  ellipse(jx,jy,100,100);
  if(mousePressed) {
    check_base();
    check_joystick();
  }
  float angle1 = map(base,100,500,1,179);
  float drive = map(jy,100,500,1,-1);
  float steer = map(jx,300,700,-1,1);
  float left = (drive + steer) * 255.0;
  float right = (drive – steer) * 255.0;
  left= constrain(left, -255,255);
  right = constrain(right,-255,255);
  left = map(left,-255,255,0,255);
  right = map(right,-255,255,0,255);
  cmd[1] = (byte)left;
  cmd[2] = (byte)right;
  cmd[3] = (byte)base;
  c.write(cmd);
}
```

Programming the Robot

- Now that the graphical user interface has been written, it is time to write the program for the robot to receive the command packet and process it.
- The code to initialize should be familiar as they were used in previous sections.

Robot Program

- The program starts with the same include files as in the first WiFi lesson. The only extra include file is for the servo so the robotic arms can be controlled.

```
#include <Servo.h>
#include <ESP8266WiFi.h>

WiFiClient client;
WiFiServer server(80);
```

Robot Program

- Create servo objects for each joint of the robotic arm.
- Create the char array for the command packet.

```
#include <Servo.h>
#include <ESP8266WiFi.h>

WiFiClient client;
WiFiServer server(80);

Servo base;

unsigned char cmd[4];
```

Robot Program

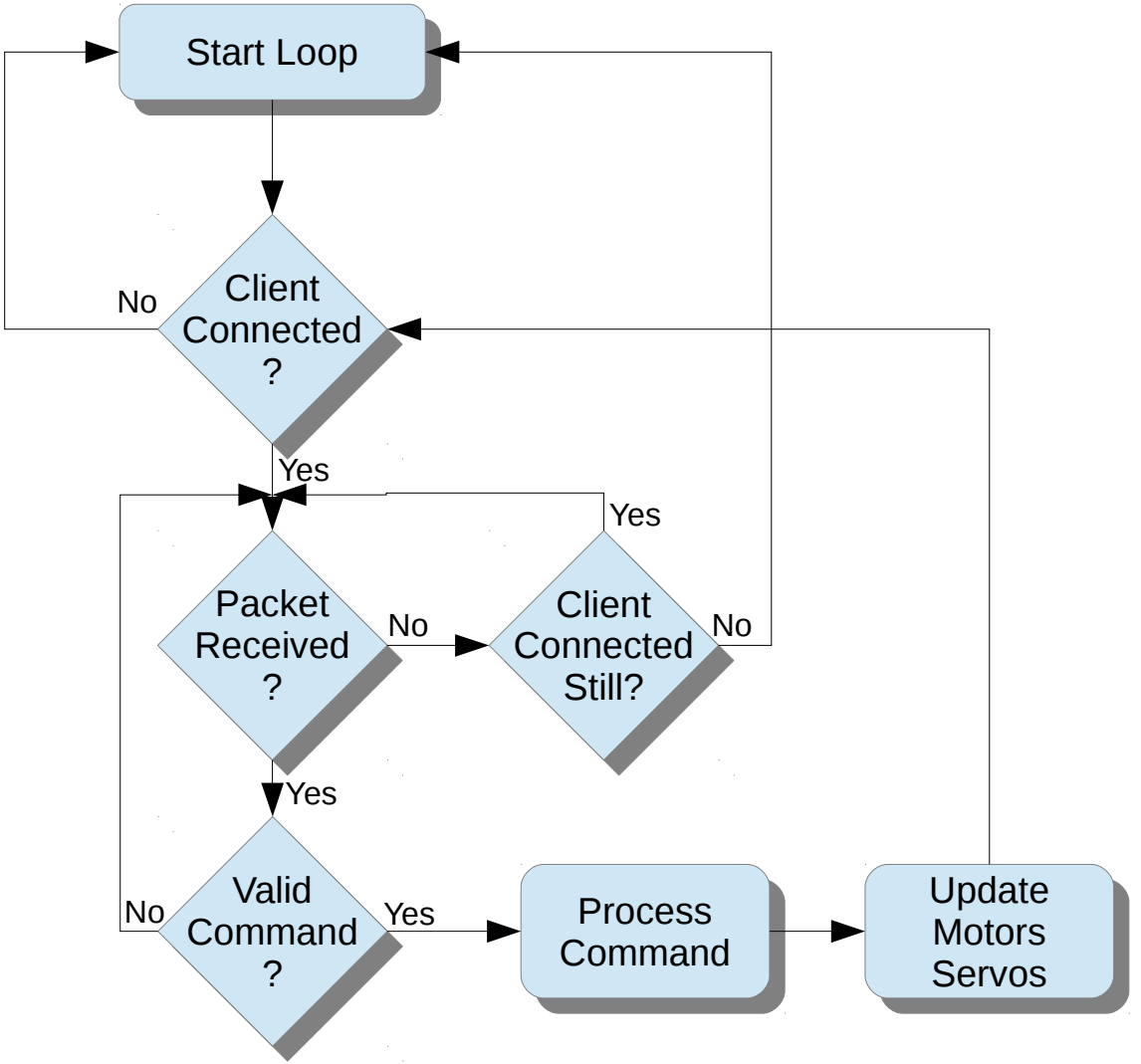
- Initialize the four pins for controlling the wheels.
- Set up the WiFi to operate as an access point.
- Set up the three ports to operate as servo controllers.
- Preposition the robotic arm into a safe position.
- This completes the initialization. Next is the **loop()** function that is added after the **setup()** function.

```
void setup() {
  Serial.begin(115200);
  pinMode(13,OUTPUT);
  pinMode(14,OUTPUT);
  pinMode(15,OUTPUT);
  pinMode(16,OUTPUT);
  WiFi.mode(WIFI_AP);
  WiFi.softAP("myrover");
  server.begin();
  base.attach(0);

  base.write(90);
}
```

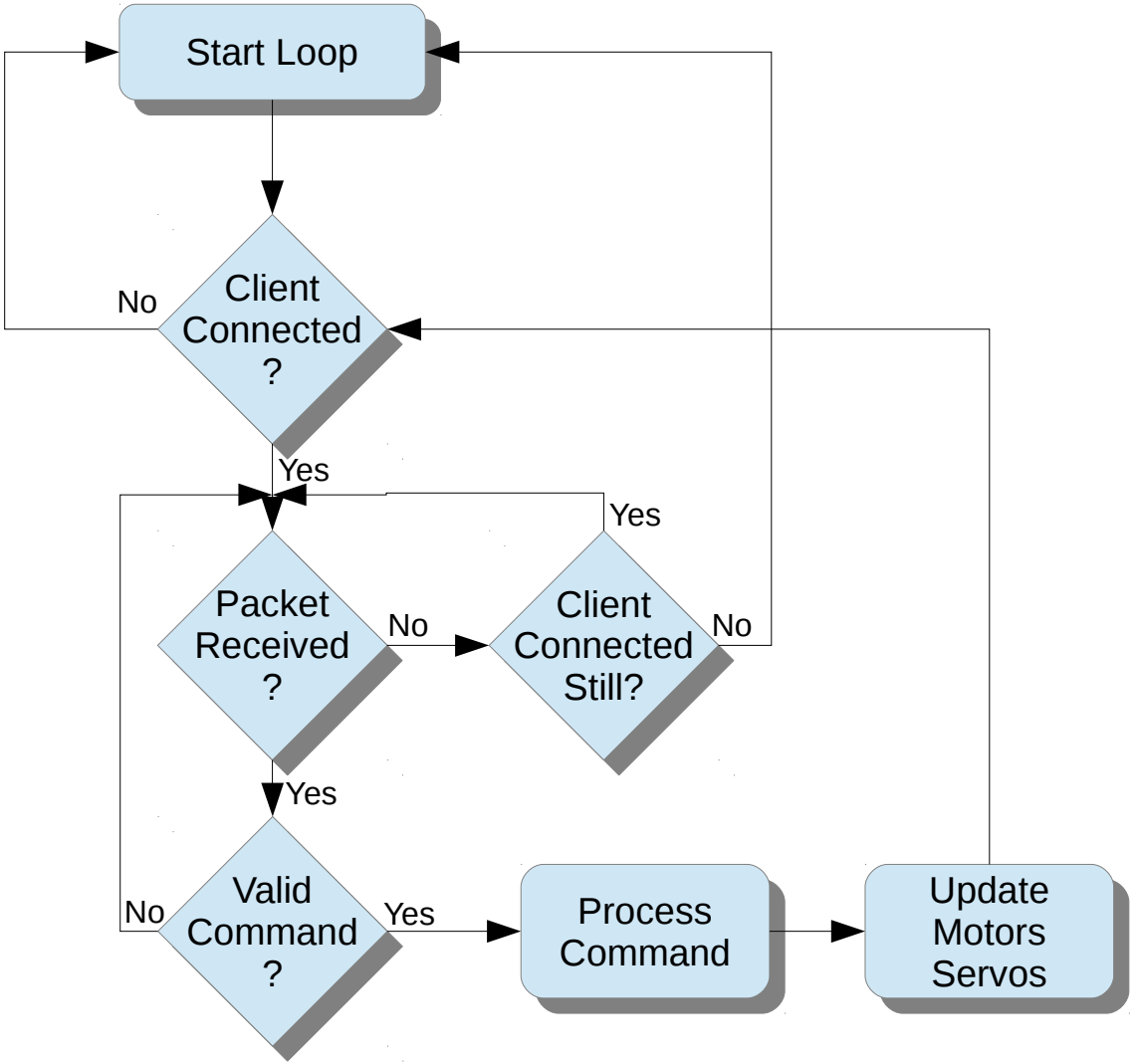
Robot Loop()

- In this section, the loop function will do multiple things.
- It will check to see if a client (laptop) has connected.
- It will then look for a command packet.
- Once a command packet is received, it will process it and update the state of the motors and servos.
- The flow chart to the right shows the code flow.



Robot Loop()

- The flow chart helps show how the program operates. The diamonds are decisions the program makes using commands like **if()**. The rectangles are the actions taken such as adjusting the servo angles and the speed of the motors.



Robot loop()

- First thing to do in the code is to check if a client has connected to the robot.
- First, the server.available() function returns a value indicating if a client has connected. If not, a zero is returned. If so a number is returned assigned to the client. Multiple clients can connect and each one gets a unique number.

```
void loop() {
  client = server.available();
```

Robot loop()

- The check for a connected client is a simple **if()** statement.
- You notice a comparison statement is not used in the **if()** statement. The variable client can be used to determine if the result is true or false. False is indicated by the value zero. Any other value is considered true.
- A diagnostic print statement is included so we can tell when a client connects.

```
void loop() {
  client = server.available();
  if(client) {
    Serial.println("Connected");
```

Robot loop()

- A **while()** loop will be used to determine when the client disconnects. Again, as long as **client.connected()** does not return a zero, the **while()** loop will continue to execute.

```
void loop() {
  client = server.available();
  if(client) {
    Serial.println("Connected");
    while(client.connected()) {
```

Robot loop()

- Now there is a check for a command packet. There is another **while()** loop waiting for data to be sent from the client. This **while()** loop also checks to see if the client disconnected. If this check does not occur and the client does disconnect, the while loop would never exit.
- **break** is a command to force the exit of the **while()** loop regardless of the results of the condition in the **while()** statement.

```
void loop() {
  client = server.available();
  if(client) {
    Serial.println("Connected");
    while(client.connected()) {
      while(client.available() == 0) {
        if(!client.connected()) break;
        delay(1);
      }
```

Robot loop()

- Once data is available, one byte is read and checked if it is the heading for the command packet. The heading byte is **C**.

```
void loop() {
  client = server.available();
  if(client) {
    Serial.println("Connected");
    while(client.connected()) {
      while(client.available() == 0) {
        if(!client.connected()) break;
        delay(1);
      }
      char a = client.read();
      if(a == 'C') {
```

Robot loop()

- If the heading is correct, the rest of the command packet is read into **cmd** array.
- function readBytes(var,num); simplifies getting the whole command. It moves a specified number of bytes into an array.

```
void loop() {
  client = server.available();
  if(client) {
    Serial.println("Connected");
    while(client.connected()) {
      while(client.available() == 0) {
        if(!client.connected()) break;
        delay(1);
      }
      char a = client.read();
      if(a == 'C') {
        client.readBytes(cmd,3);
```

Robot loop()

- Now, the command bytes are processed.
- The servo positions are updated for each joint.

```
void loop() {
  client = server.available();
  if(client) {
    Serial.println("Connected");
    while(client.connected()) {
      while(client.available() == 0) {
        if(!client.connected()) break;
        delay(1);
      }
      char a = client.read();
      if(a == 'C') {
        client.readBytes(cmd,3);
        base.write(cmd[2]);
```

Robot loop()

- The speed of the motors are mapped from the range of 0 to 255 to a range of -1023 to 1023.
- The PWM range for the motors is 0 to 1023.
- The negative numbers will be used for direction control.

```
void loop() {
  client = server.available();
  if(client) {
    Serial.println("Connected");
    while(client.connected()) {
      while(client.available() == 0) {
        if(!client.connected()) break;
        delay(1);
      }
      char a = client.read();
      if(a == 'C') {
        client.readBytes(cmd,3);
        base.write(cmd[2]);
        int left = map(cmd[0],0,255,-1023,1023);
        int right = map(cmd[1],0,255,-1023,1023);
```

Robot loop()

- Now, the motor speed is set.
- First the direction is checked. If the value is positive, the motors are set to the speed by setting the duty cycle of one digital port to the value and the other to zero.

```
        int left = map(cmd[0],0,255,-1023,1023);
        int right = map(cmd[1],0,255,-1023,1023);
        if(left > 0) {
          analogWrite(15,left);
          analogWrite(16,0);
        } else {
          analogWrite(16,-left);
          analogWrite(15,0);
        }
        if(right > 0) {
          analogWrite(13,right);
          analogWrite(14,0);
        } else {
          analogWrite(14,-right);
          analogWrite(13,0);
        }
      }
    }
    Serial.println("Disconnected");
  }
}
```

Robot loop()

- If the value is negative, the opposite digital port is set to the negative of the negative value which makes it positive. The other digital port is set to zero.
- Afterwards, the rest of the code is close brackets to close out the **while()** loops and **if()** statements.
- The print statement at the end lets you know if the client disconnected.

```
        int left = map(cmd[0],0,255,-1023,1023);
        int right = map(cmd[1],0,255,-1023,1023);
        if(left > 0) {
          analogWrite(15,left);
          analogWrite(16,0);
        } else {
          analogWrite(16,-left);
          analogWrite(15,0);
        }
        if(right > 0) {
          analogWrite(13,right);
          analogWrite(14,0);
        } else {
          analogWrite(14,-right);
          analogWrite(13,0);
        }
      }
    }
    Serial.println("Disconnected");
  }
}
```

Testing

- To test, upload the robot code into the robot processor.
- Connect the laptop to the Robot access point.
- Once connected, run the GUI program.
- Operate the robot.
- If some things do not work, insert Serial.println() statements in the robot code to print out variables to see if commands are formatted properly.

End